August 1982

Report No. STAN-CS-82-924

(12

Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations

by

Marsha J. Berger

Department of Computer Science

Stanford University
Stanford, CA 94305

ADAPTIVE MESH REFINEMENT

FOR HYPERBOLIC PARTIAL DIFFERENTIAL EQUATIONS

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

N00014-75-C-1132

by

Marsha J. Berger

August 1982

Adaptive Mesh Refinement

for Hyperbolic Partial Differential Equations

Marsha J. Berger, Ph.D.
Stanford University, 1982

In many time dependent simulations, the solution on most of the domain will be fairly smooth, with discontinuities or highly oscillatory phenomena occurring over only a small fraction of the domain. In problems such as these, a mesh refinement approach can be the most efficient, and often the only practical, solution method. Refined grids with smaller and smaller mesh spacing are placed only where they are needed. Since we are solving a time dependent problem, the regions needing refinement will change, and therefore our grids must adapt with time as well.

This thesis presents a method based on the idea of multiple, component grids for the solution of hyperbolic partial differential equations (pde) using explicit finite difference techniques. Based upon Richardson-type estimates of the local truncation error, refined grids are created or existing ones removed to attain a given accuracy for a minimum amount of work. In addition, this approach is recursive in that fine grids can themselves contain even finer subgrids. Those grids with finer mesh width in space will also have a smaller mesh width in time, making this is a mesh refinement algorithm in time and space.

In chapter 2 we present the full mesh refinement algorithm. This includes a discussion of the error estimation procedure, as well as the integration algorithm itself. We calculate how often the error estimation and subsequent grid generation should be performed to minimize the total cost of the algorithm.

In chapter 3 we discuss the interface equations used at the boundary of the fine and coarse grids. Boundary schemes are presented for those component grids with boundaries internal to the problem domain, since in these cases boundary conditions are not supplied with the pde. We present a new stability proof for the case of interpolation interface conditions with the Lax-Wendroff difference scheme for mesh refinement in time and space.

We also include in this chapter a discussion of conservation properties of the algorithm, including both the interface conditions and the regridding procedure. We develop a procedure to derive conservative interface conditions so that if the approximate solutions converge, they will converge to a weak solution of the pde. This is important in computing the correct shock location in cases where the solution is discontinuous.

In developing this algorithm, we have drawn heavily from many areas of computer science not usually found in numerical analysis applications. This is especially evident in the technique we use to create refined subgrids. Our subgrids are rectangles rotated to best fit the area of the domain needing refinement. The rotation allows us to have grids whose coordinate systems are normal and tangent to a discontinuity in the solution. Chapter 4 presents the clustering and grid generation algorithms we use to create the fine grids. Chapter 5 presents the data structures we use to handle the big data management problem found in adaptive methods for pde. The efficiency of our method is due in large part to these two components.

We have begun to extend this method to the solution of steady state problems. This is described in chapter 6. Chapter 7 contains several numerical examples in both one and two space dimensions. We summarize in chapter 8 with conclusions and directions for future research.

Approved for publication:

By _____
For Major Department

By _____
Dean of Graduate Studies & Research

Acknowledgments

I am happy to thank my advisor, Joe Oliger, for suggesting and guiding what has turned out to be such a fruitful topic of research. Gerry Hedstrom, a member of my reading committee from Lawrence Livermore Laboratory, and Garry Rodrigue, also from Lawrence Livermore Laboratory, have made many helpful suggestions and been a pleasure to work with during the last few years. It is a pleasure to thank Gene Golub for his enthusiastic efforts to make the numerical analysis group so lively and exciting here at Stanford.

Many of the students at Stanford have contributed in many ways to the work in this thesis. Randy LeVeque and Lloyd Trefethen have been good friends as well as colleagues in the study of the numerical solution of pdes. Jonathan Goodman has made many excellent suggestions during the course of this work. In particular, thanks are due to him for suggesting a proof of the identity (3.2.11) in section 3.2. I thank Bill Gropp for many helpful discussions about mesh refinement. Most of the graphics in section 7 are due to him, as well as some of the initial coding of the the 2-D mesh refinement program. Doug Baxter collaborated in much of the testing and development of the clustering algorithms in section 4.1, both in his masters thesis, and for a long time afterwards. There are several other people at Stanford I would like to thank— John Bolstad, Chris Fraley, Wei-Pai Tang, Dan Boley, Janet Wright, Rich Pattis, Peter Nye and Rob Schreiber, who have helped in many ways during my years at Stanford.

I especially thank Amy Lansky, Anita Mayo, Russ Caflisch, and Jonathan Goodman for their friendship and support. They have helped make my life at Stanford happy as well as productive.

This work was supported in part by the Office of Naval Research under contract N00014-75-C-1132, and by the National Science Foundation under grant MCS-7702082. Computer time for this work was provided by the Stanford Linear Accelerator Center of the U.S. Department of Energy. The manuscript was produced using TEX, the computer typesetting system created by Donald Knuth at Stanford.

Table of Contents

Chapter 1: INTRODUCTION

1.1 Problem

This thesis presents a numerical method for efficiently solving a class of partial differential equations (pde) in one or more space dimensions. The method we use is adaptive mesh refinement. It is suited for those equations where the solution is highly oscillatory, discontinuous, or irregular in some other way in only a small portion of the domain. We give two examples to illustrate these phenomena and to motivate our adaptive approach. The scalar wave equation

$$u_t = -u_x$$

is the simplest linear hyperbolic pde. If the initial conditions consist of a single jump

$$u(x, t = 0) = \begin{cases} 1, & x < 0 \\ 0, & x > 0 \end{cases}$$

the solution is a step function where the discontinuity propagates to the right along the line $x = t$. On either side of the discontinuity the solution is constant.

For another example, consider the inviscid Burgers equation

$$u_t = uu_x \tag{1.1}$$

for $0 \leq x \leq 1$ with initial conditions

$$u(x, 0) = \cos(3\pi x)$$

and boundary conditions

$$u(0) = 1,\ u(1) = -1.$$

Although we start with very smooth initial conditions, the solution to this pde becomes an N-wave (see figure 1.1(b)) and finally reaches the steady state solution of a single jump discontinuty (see figure 1.1(d)). In problems like this, the location (or even the number) of discontinuities are often not known in advance, and in any case, it changes throughout the computation.

This type of solution occurs in more realistic problems as well. In aerodynamical calculations around an airfoil, if the free stream Mach number is near the speed of sound, a typical steady state solution has a supersonic bubble ending in a shock above the airfoil, with a smooth flow in the elliptic region (Morawetz, [1982]). It is problems such as these that can benefit from the solution algorithm described in this dissertation.

For any given computation, one would like to set up a grid that minimizes the discretization error and maximizes the computational efficiency of the solution method. For some of the three dimensional calculations done today, even the finest grids possible provide inadequate resolution for some features of the solution. In both memory and time requirements, these computations are exhausting modern computing resources (Levine, [1982]; Ushimaru, [1982]). It is necessary then to have some mechanism which puts the grid points where they are needed most. Since it is often not known in advance just where that is, it is necessary for the algorithm to be able to figure it out as the solution is computed.

In this dissertation we present a numerical method based on local grid refinements in both space and time for efficiently solving equations whose solutions are locally irregular. Our approach is to generate as many independent, refined subgrids as needed in the irregular region of the domain. The subgrids we generate are rectangles in the coordinate system of the original coarser grid. The solution on each subgrid can then be approximated by standard finite difference techniques, as done on the original coarse grid. Since we are solving a time dependent problem, the irregular regions will change in time, and thus our grids must adapt in time in response to the solution. Our algorithm is very general, and it makes no assumptions about the number or type of these irregular regions, nor about their direction of motion.

We give a simple example in one dimension to illustate our adaptive approach. We will integrate equation (1.1) using the Lax-Wendroff difference scheme. If we use a spatial step size of $h = .025$, or 40 points in the interval [0,1], as we have in figure 1.1(a), the initial sine waves are perfectly well resolved. After four time steps with $\Delta t = .0125$, our algorithm decides two new subgrids are needed. These new grids have a space step 1/4 that of the original (underlying) grid. Several steps later, another level of refinement is added, so that the mesh spacing on the new grids is now 1/16 the original mesh width. The solution becomes an N-wave, with the grids illustrated underneath the plot in figure

1.1(b). As the solution progresses, the rarefaction is eaten by the shocks, the shock fronts advance, and the grids move (figure 1.2(c)). The final steady state solution and grid structure is in figure 1.1(d). By examining the solution, in particular by estimating the local truncation error in the solution, we create new grids where the error in the solution is greatest. Following Pereyra and Sewell [1975] we try to *equidistribute* the error, so that the minimum number of grid points is used to obtain a given level of accuracy on the entire grid.

Most of the pdes we solve are hyperbolic equations. Since the solution of a hyperbolic pde can be characterized as containing waves which have a finite speed of propagation, we can move the fine grids often enough so that the regions needing refinement don't escape the fine grids. This holds if we add small viscous terms, or compute steady state solutions, and there is no reason in principle that we can't extend our method to parabolic pdes as well.

We mention here one other aspect of our multiple grid structure. In any problem, a regular grid geometry is much preferred for several reasons. On unequally spaced grid points, it can be expensive to approximate derivatives with differences to a high order of accuracy. Secondly, in more than one space dimension, there is a big data management problem of keeping track of unevenly spaced points in an efficient way. It is much easier to generate a grid on a regular region than an irregular one. If we're not lucky enough to be working on a computational rectangle, the first difficulty above motivates the generation of what are called *body-fitted coordinates*, that is, a coordinate system transformation from the physical domain to a rectangle. This makes the boundary conditions easy to apply, especially if the original coordinate system is orthogonal, and the transformation preserves this, *i.e.*, a conformal map. However, it is often difficult to find such a transformation for highly non-uniform regions, such as cascades of turbines. In addition, it can be difficult to control the mesh spacing over the entire region. For example, a map from a highly cambered airfoil to a circle may have enough grid points for good resolution on the boundary of the airfoil, but not enough under the airfoil. Recently, the alternative approach in these situations is being considered of using several overlapping component grids to discretize the domain (Eiseman, [1979]; Starius, [1980]; Kreiss, [1982]). The computations that must be done on refined subgrids are the same as

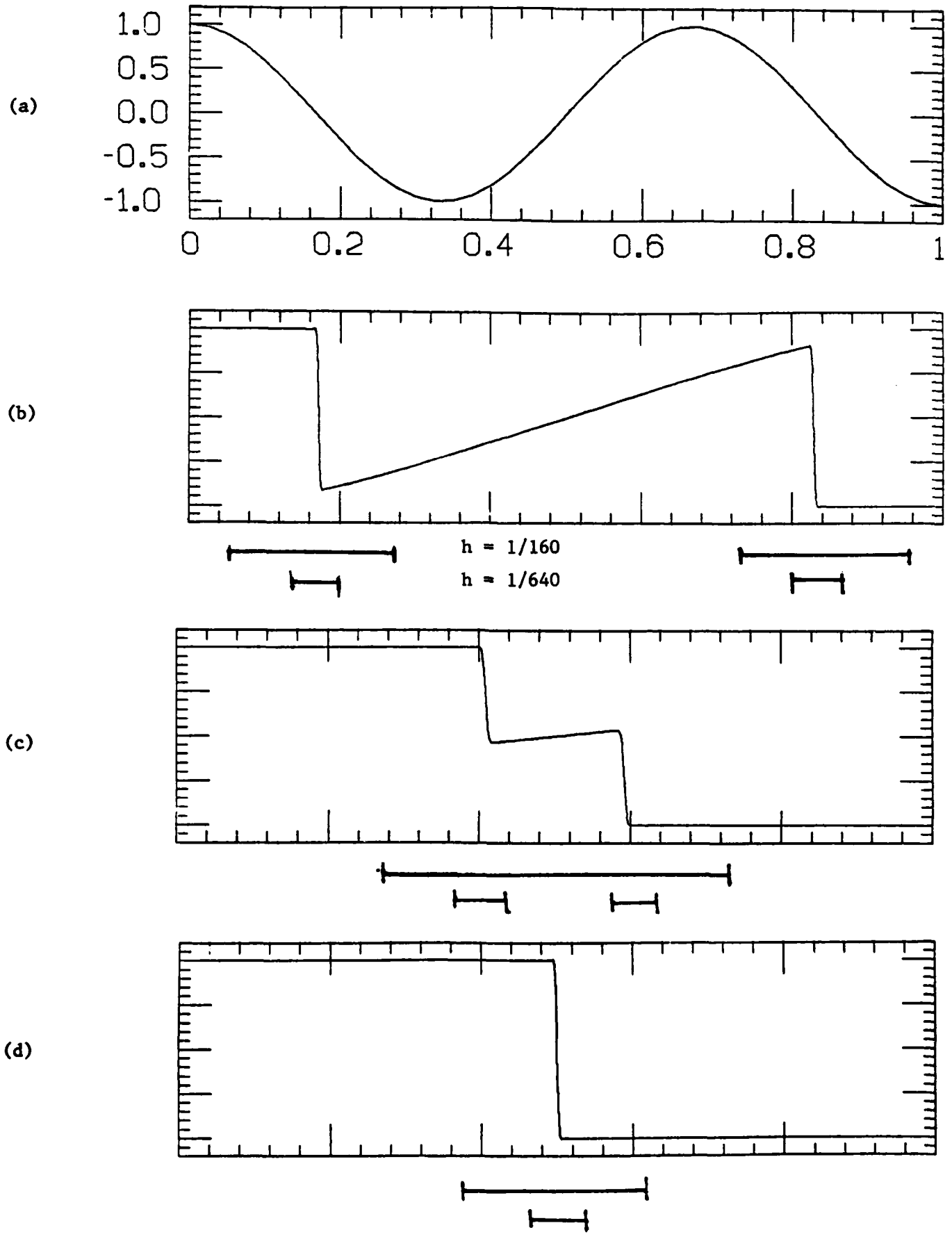


Figure 1.1

those done on independent component grids. In this thesis we present a general enough framework to compute solutions on both of these types of grid structures.

The rest of this chapter presents a survey of adaptive methods in pdes, where there is now a lot of research activity, and a brief history of the use of component grids, which has received very little attention up to now. In chapter 2, we describe the basic mesh refinement algorithm. We start with a description of our grid structure. In our method we use rotated rectangles for the subgrid geometry. In addition, this algorithm is recursive in that subgrids with finer and finer mesh width can themselves be nested in other subgrids. We then describe the integration algorithm and the grid-to-grid communication routines needed for this kind of grid structure. In this chapter we also describe the algorithm we use to decide where the fine subgrids are needed. We base the refinement decision on estimates of the local truncation error. We estimate this using a very efficient, automatic procedure based on Richardson extrapolation.

Chapter 3 is devoted to the interface conditions we use at the boundary of the fine and coarse grids. In section 3.1 we describe all the boundary condition algorithms. In section 3.2 we present a new stability proof for one of the interface conditions for mesh refinement in time and space. In section 3.3, we discuss conservation properties of the algorithm. Conservation is important in computing the correct shock location of a discontinuous solution. Although our discussion is so far restricted to one space dimension, many of the results apply in more than one dimension. In one dimension, exact conservation is not critical, since if there is a discontinuity in the solution, there should always be a fine grid around it. However in two dimensions, because of the way our grids are designed, a shock can cross subgrid boundaries, and conservation properties of the algorithm become important. In this section, we show how to derive a general class of conservative interface conditions which will converge to weak solutions of the pde if the approximations themselves converge.

Most of the non-numerical computer science material is presented in chapters 4 and 5. Chapter 4 contains the algorithm we use to generate subgrids. First we identify and characterize those areas needing refinement, drawing on material from the pattern recognition literature. Then we determine the orientation of the rectangular subgrid. The efficiency of this algorithm is fundamental to the success of the mesh refinement algorithm. In Chapter 5 we describe the data structures we use to keep track of the

adaptive grid structure during a computation. This turns out to be surprisingly easy, but again fundamental to the overall success of the algorithm. We use the fact that a fine grid is nested in a coarser grid to determine a tree data structure where each node represents a grid, and make the correspondence between a parent node and a coarser, parent grid.

We have begun to apply this method to computing steady state solutions. Our goal is to compute steady state transonic flows about an airfoil in two or even three dimensions. We briefly describe in chapter 6 some of the modifications of the algorithm for this type of problem, and we show some preliminary results for our one dimensional steady state model problem.

Chapter 7 contains computational examples for this method. We do both linear and nonlinear, one and two dimensional computations. We give a short description of the parameters that must be set to run our mesh refinement code, to show that this method really is automatic and therefore feasible.

1.2 Adaptive Methods

Adaptive mesh refinement and adaptive mesh generation have become much more common among numerical methods for solving pde's. Adaptive techniques have been used for other problems for a long time: adaptive quadrature routines, (deBoor, [1971]), and variable-step variable-order ODE integrators (Shampine and Gordon, [1975]) are examples of this. Almost all approaches to solving two point boundary value problems are adaptive in the sense of iterating between specifying a mesh, solving on that mesh, adapting the mesh to reflect the latest solution, re-solving, and so on, (Kreiss and Kreiss, [1981]; Lentini and Pereyra, [1977]).

Closer to our problem are the adaptively refining elliptic pde solvers (Babuška and Rheinboldt, [1978]; Bank, [1981]). These adaptive solvers all use finite element methods. Grid refinement is based on refinement of an individual element, usually a triangle or rectangle. These methods are very successful, and extensive theory has been developed for them. Recently, adaptive finite element methods have started appearing for parabolic pdes as well (Davis and Flaherty, [1982]; Gannon, [1980]; Sherman and Seager, [1981]). Some of the initial ideas for adaptive mesh refinement using finite differences were first suggested several years ago by Oliger [1979]. However, until recently, adaptive finite element solvers have received much more attention in the literature than finite difference methods, which are more commonly used for solving hyperbolic problems.

Non-adaptive mesh refinement techniques have been used for a long time in solving hyperbolic pdes. This is particularly evident in numerical weather forecasting models. The two most common approaches to this are surveyed by Elvius and Sundström [1979] and are termed *two way interaction models*, where grid point density varies either continuously or stepwise, and *one way interaction models*, where a fine grid is used on a limited domain, getting its boundary models from a global grid with a crude mesh. Among other problems, these methods suffer from the lack of adaptivity. Browning, Kreiss and Oliger [1973] show how poor the results can be when high frequency waves escape to the coarse grid where they are poorly resolved.

In problems where it is known in advance where grid refinement is necessary, it is better and less expensive to put it in once and for all than to do it adaptively. This is done for example in codes that calculate flow about an airfoil (Steger and Chaussee,

[1980]). The computational mesh is refined in the boundary layer, and this is built directly into the coordinate system.

Since it is often difficult to predict the solution well enough to know where the grid refinement is needed, it is clear that the next step is to make the grid adaptation dynamic. Most methods today for adaptive mesh refinement for hyperbolic pdes fall into two main categories. We separate methods into *deformed mesh* methods, and *regular refinement* methods. In deformed mesh methods, we include any method which distributes grid points in a non-uniform way. For example, in the Moving Finite Element method, (K. Miller *et al*, [1981]), the location of the grid points is determined jointly with the solution itself by minimizing the residual of the finite element equations. In more than one space dimension this can lead to a distorted, irregular mesh. This is also the case with Lagrangian methods. Briefly, Lagrangian methods associate a grid point with a moving material particle. In problems involving different material substances, these methods have great advantages (Noh, [1976]). It is typical in Lagrangian calculations to take a remap step of interpolating back onto a fixed grid after some number of steps, in order to prevent the grid distortion. If this is not done, grids can get so distorted that it is difficult to determine neighboring grid points, if in fact they are still well-defined. Not to be too pessimistic, these methods are excellent front tracking methods. They follow discontinuities and keep them sharp. This is basically the approach used by Rai and Anderson [1981] in their procedure for aligning a coordinate line with a shock front. However, it seems that these approaches would have difficulty with several intersecting fronts.

The other common approach to adaptive mesh refinement is what we have called regular refinement methods. This is the approach used by Dwyer *et al*, [1980] and Winkler [1976] for two examples. In their problems, the grid refinement is done adaptively in only one dimension. Using their refinement criteria, be it equidistributing the gradient of the solution or something more complicated, the mesh spacing in a coordinate direction is determined for the entire domain. In common approaches to two-dimensional mesh refinement, the simplest techniques have the drawback illustated in figure 1.2. To resolve the region indicated by the circle, a large area must be unnecessarily refined. This approach is also common in transonic flow calculations where grid lines are clustered

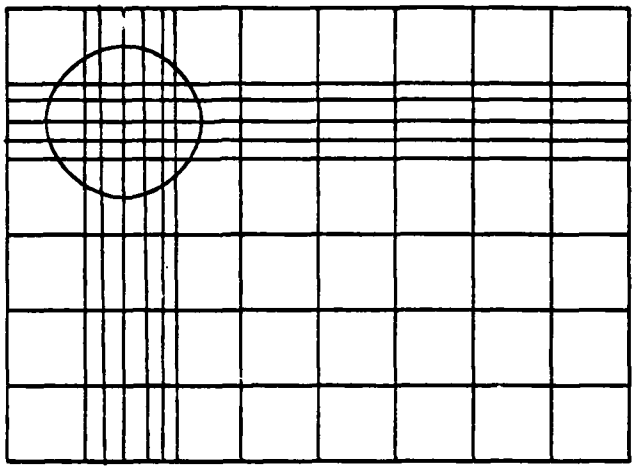


Figure 1.2 Uniform Refinement

near the leading and trailing edges of the airfoil and near the location of the shock (Nakamura and Holst, [1981]; Ushimaru, [1982]).

This was partially alleviated in the work of Gropp [1980], who used the idea of local grid refinement for each row between specified columns. In his work, several refined grids may be specified in a single row. This is the same type of approach used by multigrid practitioners (Brandt, [1977]), except that their data structure limits them to one refined region per row. This method has storage overhead for each refined row. It still has unnecessarily refined portions of the grid if there is a smooth region between the first and last column to be refined. Bolstad [1982] has overcome some of these difficulties in his use of local nested refinements in one dimension. His work is the most similar to the work presented in this thesis. Most adaptive methods have the restriction that they start with a fixed number of points which they distribute according to some criteria. It is advantageous to also be able to add grid points or remove them during the calculation. The methods of Gropp and Bolstad do not have this limitation.

We have not been exhaustive in outlining approaches to adaptive grid refinement, but have only sketched some approaches to put this method in context. In addition, methods can be considered adaptive in a totally different sense. For example, in hybrid methods the solution is computed using two different schemes, and a different linear

combination of them is used at each point depending on the solution properties at that point. Another type of adaptivity is evident in multigrid methods (Brandt, [1977]; Foerster, Stüben and Trottenberg [1981]). This method uses a nested sequence of grids with finer and finer mesh spacing to solve a problem by iteration. In this method, the rate of convergence is monitored during each iteration of the solution. If it is slow, a related iteration proceeds for a while on a coarser grid. Thus, the structure of the algorithm itself is adaptive. However, we do not discuss these other types of adaptivity further.

1.3 Component Grids

There is little practical experience and even less theory to be found in the literature about the use of component grids. By component grid computations we mean computations using independently defined grids in different coordinate systems. Several authors have ways of constructing a grid which changes smoothly from one coordinate system to another (Eiseman, [1979]). There are also techniques to generate a grid which is body-fitted to several disjoint components, either by conformal mapping (Halsey, [1981]), or by solving elliptic pdes (Thompson, [1974]). In general, it would be easier to use a separate body-fitted grid for each component. The trade-off is of course the increased difficulty of interfacing the component grids. This thesis presents a general framework in which to view the computational effort, and a data structure to manage the components. In this section, we will only briefly outline two approaches to solving a problem with component grids.

Starius [1979] presents what he calls a composite mesh difference scheme for the shallow water equations. The two grids he uses are drawn schematically in figure 1.3. Starius uses a third order interpolation formula to set the interior boundary equations of the outside grid using values from the inside grid, and vice versa for the exterior boundary of the inside grid. For the interpolation to be centered, the grids must overlap by at least one mesh width. His results show that despite the interpolation, which is not a conservative procedure, the amount of water in the computation (which is conserved by the original pde) changed by less than $5 \times 10^{-3}\%$. Starius also has a stability proof in one dimension for a composite mesh difference scheme using the Lax-Wendroff difference method on both grids.

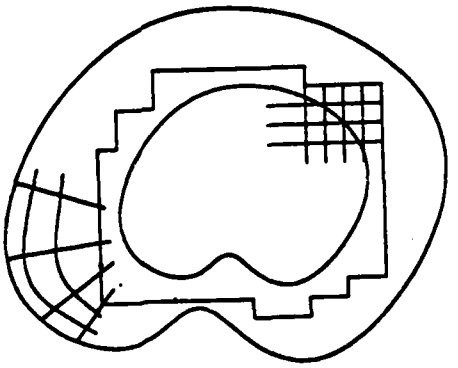


Figure 1.3 Starius-type grids

Atta [1981] has done two dimensional two component calculations for transonic flow about an airfoil. He uses a body-fitted grid around the airfoil, and a Cartesian far field grid. Strangely, he tries the non-symmetric approach of using Neumann type boundary conditions for the inner boundary of the outer grid, and Dirichlet type boundary conditions for the outside of the inner grid. The solution is obtained by alternating iterations on one grid then the other. Atta's results show that the accuracy of the interface equations depends on the amount of overlap, as does the number of iterations for convergence. There are many questions which remain to be answered about how to compute using several component grids.

Chapter 2: ALGORITHM DESCRIPTION

2.1 Grid Description

We start the presentation of the algorithm by describing the type of grids we use in solving a problem with our adaptive mesh refinement strategy. We also develop the notation and terminology needed to discuss these grids.

At the start of a computation only the coarsest, or *base* grid is specified by the user. This base grid, denoted G_0, will remain fixed for the duration of the computation. G_0 itself may be composed of several possibly overlapping component grids. We use the term *grid* to refer to the convex hull of the point set of the grid, rather than the point set itself. Thus, we say that grids overlap if their convex hulls have a nonempty intersection. We call each component grid $G_{0,j}$, and loosely say that G_0 is the union of its components $G_{0,j}$. Each component grid is required to be locally uniform in some coordinate system. They need not form a simply connected domain. In addition, these grid components do not necessarily have the same mesh width. For example, we might use a grid over a region with a boundary layer that has a much finer discretization than that of the grid covering the interior of the domain. Furthermore, within each grid the mesh spacing in the coordinate directions need not be equal. For simplicity of exposition, however, we ignore these points and assume G_0 has mesh spacing $h_x = h_y = h_0$ on all components $G_{0,j}$.

During a computation, refined subgrids will be created adaptively in response to some feature in the transient solution, such as the estimated error in the solution or the appearance of shock fronts. Our goal is to generate the subgrids to best fit the area of the domain where they are needed. The subgrids we create are rectangles of arbitrary orientation. By keeping the subgrids locally uniform, integration on these subgrids can be very efficient. By allowing the arbitrary orientation, it is possible to have a coordinate system which is locally approximately normal and tangent to some feature in the solution, for example a shock front. In addition, storage requirements can be significantly smaller by allowing rotated rectangles. Figure 2.1 illustrates a grid structure with two component

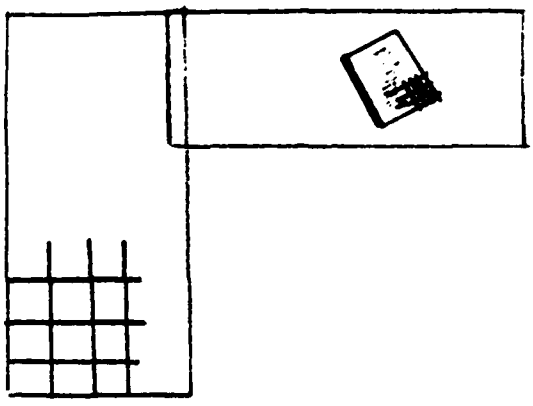


Figure 2.1

grids at the coarsest level, one of which has a refined subgrid. The darkened area in the subgrid represents the part of the coarse grid needing refinement.

It is important to realize that these subgrids are not *patched* into the coarse grid. Rather, a subgrid should be thought of as *overlaying* a coarser grid. Each grid is defined independently of the other grids, with its own solution vector, storage, *etc*. In this way, each subgrid can be integrated (almost) independently of the other grids. It also easily allows for the possibility of using moving subgrids, even if the coarsest grid is stationary. Furthermore, by keeping the grids independent, the algorithm can be viewed as a method of domain decomposition, and is well suited for multiprocessor architectures currently under development. Other authors (Simpson [1978]; Bolstad [1982]) have created refined meshes which are connected into the underlying coarse grid to make one global grid. In their approach, fine grid points are merged into the set of coarser grid points. In one dimension it is possible to do this fairly easily, as Bolstad does. In several dimensions this can be a mess. This approach destroys the local uniformity of each grid, substantially slowing down and complicating the integrator, as well as preventing its vectorization. In addition, it requires storage overhead and processing which is typically proportional to the number of refined gridpoints. An advantage of the use of overlaying grids is that it can be implemented with storage overhead proportional to the number of refined grids. This approach does need some kind of data management scheme to keep track of the

grids. This is a smaller problem than keeping track of each point, however. The data structures we use for this are presented in chapter 5.

It is possible that the fine subgrids will themselves contain even finer subgrids within their boundaries. We say that a point in one grid is contained in another grid if it lies in the convex hull of that grid. A grid is contained in another grid if all points in that grid are contained in the other. Similarly, a point of one grid is an interior (boundary) point of another grid if it lies in the interior (on the boundary) of the convex hull of the other grid. We define the *level* of a grid to be the depth of this nesting, that is, the number of coarser grids the fine grid is contained in. We say that the coarse grid G_0 is at *level* 0 in the grid hierarchy. The subgrids of G_0 are part of G_1 and they are said to be level 1 refinements. Refined grids within the G_1 grids are at level 2, denoted G_2, and so on. In this way, a nested sequence of grids with finer and finer discretizations may be created over some portion of the spatial domain. Each such grid is one grid component, denoted $G_{l,j}$, of G_l, where G_l consists of those grids at level l in the hierarchy having mesh width h_l. A point in the problem domain may therefore be interior to several grids, but the approximate solution at that point is defined by interpolation from the finest grid to which that point is interior.

In practice, we assume a set of possible mesh discretizations $\{h_0, h_1, h_2, \ldots, h_{\max}\}$ has been specified in advance. Each h_l is an integral multiple of h_{l+1}: a factor of four is typical. A good choice for this refinement ratio will depend on how much of the domain needs what amount of refinement. If only one part of the domain needs to be in a fine grid with mesh width $h_f = h_0/r$, it is more efficient to create one level of refinement with $h_1 = h_0/r$ than two levels each with a ratio of $\sqrt{r}$. In general, however, not all areas needing refinement will need the same amount of refinement, arguing for smaller values of the refinement ratio r. Since there is some overhead associated with refined grids, we prefer a refinement ratio of 4 over the ratio of 2 typically used with multi-grid methods. In special cases where it is expected that all areas needing refinement will need a lot of it, higher values of r can be used efficiently. We have used values of $r = 10$ in some computations. In his thesis, Gropp [1981] presents a calculation using two grid levels where $h_f = h_c^2$. He uses $h_c = .01$, which is equivalent to a refinement ratio of $r = 100$. This was done to achieve a uniform convergence rate over the entire domain, which included a region with a discontinuity, surrounded by regions where the

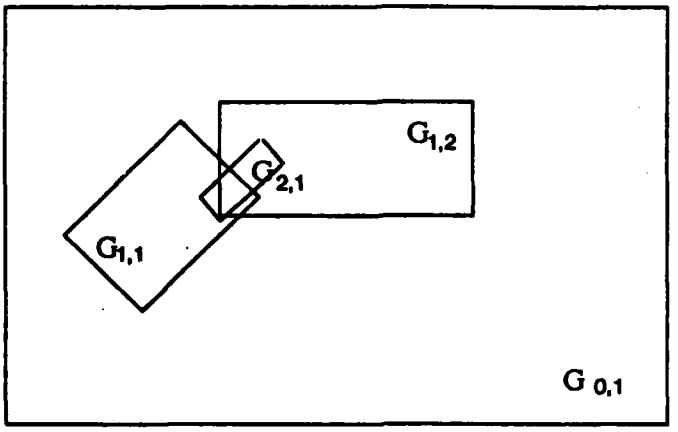


Figure 2.2

solution was smooth. In chapter 7 we present some computational experiments involving this parameter.

We emphasize an assumption we make about the grid hierarchy. If at some point during the computation there is a grid with mesh width h_l, we require that it be contained in a grid at level $l-1$, which in turn is required to be contained in a level $l-2$ grid, and so on to the coarsest level. *All intermediate grids between the finest and coarsest are be maintained.* Furthermore, each point in a fine grid at level l must be in the interior, not just on the boundary, of a grid at the next coarser level, unless it is on the physical boundary of the domain.

We now point out that not all points in a fine grid are interior to the *same* coarse grid. We call this type of nesting *level nesting*. Figure 2.2 illustrates how the grid at level 2 is nested in the union of grids $G_{1,1}$ and $G_{1,2}$ at level 1. Figure 2.2 also illustrates the complication in 2 or more dimensions of overlapping refined grids, which will come up later.

In one dimension, where a grid is just an interval, we avoid both of these situations. Level nesting in one dimension is identical to straightforward grid nesting, and overlapping of fine grids does not occur. Grids at the same level of refinement with the same mesh width are either disjoint or else they are merged into one grid spanning a larger interval.

In this way, using the ideas of independent component grids and recursive refinement, a hierarchy of nested grids is formed. The complete grid structure is denoted

$$G = \bigcup G_l,$$

where the grid structure at level l is the union of rectangular components

$$G_l = \bigcup G_{l,j}.$$

2.2 Integration Algorithm

In this section we describe the integration algorithm that we use to solve a hyperbolic pde using mesh refinement. There are three main components in this algorithm. They are (*i*) the actual time integration using finite differences that is done on each grid, (*ii*) the error estimation and subsequent grid generation, and (*iii*) the special grid-to-grid operations that must be done every time step during the integration that arise because of mesh refinement itself. We describe these three components in turn.

Most of the computational effort in our mesh refinement program is devoted to the time integration of the solution. Since each grid is defined as an independent computational entity, with its own solution vector, each grid can be advanced independently of the other grids, except for the determination of its boundary values. We must then consider the question of which grids to integrate when, and determine the order of their integration. This is made easy however by the following requirement.

Recall from §2.1 that in our grid formulation, the mesh widths h_l of grids at level l are an integral factor r of the mesh width h_{l-1} of the next coarser level. We use the same factor to set the time step on the level l grids, $k_l = k_{l-1}/r$. This is an appropriate time step for hyperbolic equations. In this way we keep the mesh ratio λ of time step to space step constant on all grids. This makes our algorithm one of mesh refinement in time and space. One of the main reasons our method is efficient is because the overly restrictive small time step of the finest grid is not applied over the entire domain.

This constant mesh ratio λ makes it easy to determine the order of grid integration. If r is the refinement ratio, for every 1 time step taken on level 0 grids, take r time steps on level 1 grids, r^2 on level 2 grids, *etc*. These steps are interleaved so that before advancing a grid, all its subgrids are integrated to the same time. At every coarse grid step, all grids should be at the same time. One coarse grid cycle is then the basic unit of the algorithm.

Figure 2.3 illustrates this in one space dimension and time. We depict one coarse grid, $G_{0,1}$, one fine grid $G_{1,1}$ at level 1, and a finer grid at level 2, $G_{2,1}$, all with the refinement ratio $r = 2$. The order of their integration, from coarsest to finest, for one coarse grid cycle is

$$G_{0,1}\, G_{1,1}\, G_{2,1}\, G_{2,1}\, G_{1,1}\, G_{2,1}\, G_{2,1}$$

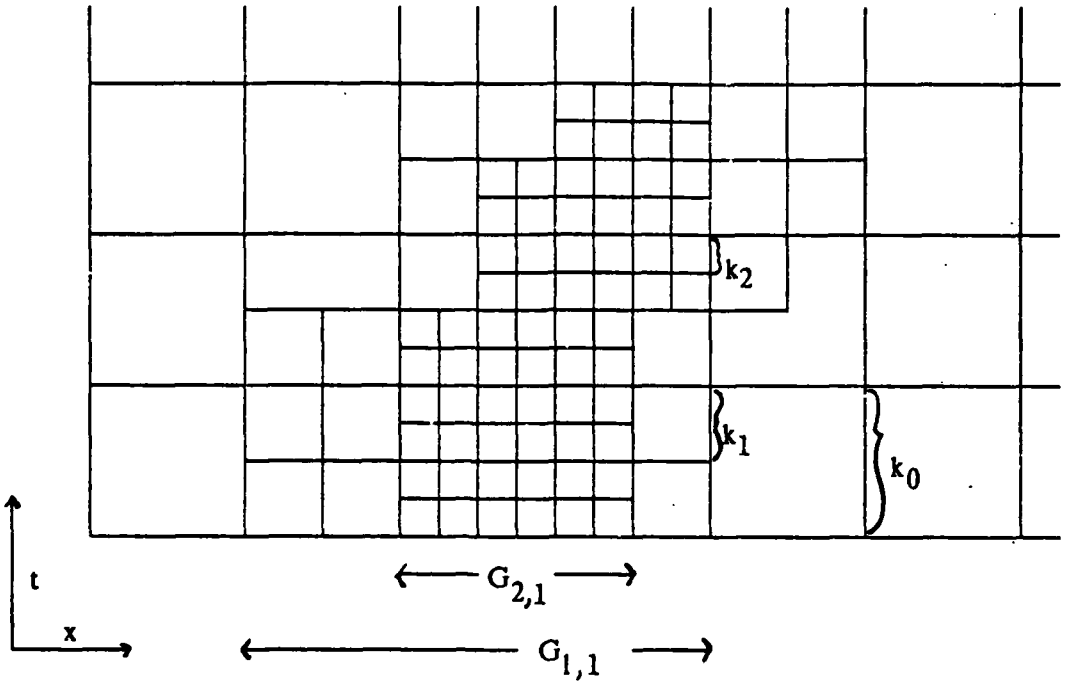


Figure 2.3

It is also possible to order the grid integration from finest to coarsest, and still maintain the principle of grids waiting for each other to catch up to the same time before continuing. If we start with the finest grid in figure 2.3, the integration order for one coarse grid cycle is exactly reversed,

$$G_{2,1}\, G_{2,1}\, G_{1,1}\, G_{2,1}\, G_{2,1}\, G_{1,1}\, G_{0,1} \quad .$$

The difference between the two orderings will be important only as an implementation detail in the choice of interface conditions. This will be pointed out later.

We make some last comments to finish discussing the integration algorithm. Since refined grids are rotated rectangles, the difference equations must be transformed into the rotated coordinate system. We would like to do this in as automatic a way as possible so that the integrator, which is supplied by the user, can be separated from the adaptive mesh routines. Since we use standard finite difference equations to integrate the pde, this is easy to do. For example, to solve

$$u_t = Au_x + Bu_y$$

in the rotated coordinate system

$$\begin{pmatrix} r \\ s \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

we can solve

$$u_t = (Ar_x + Br_y)u_r + (As_x + Bs_y)u_s,$$

or instead, as sometimes preferred,

$$u_t = \frac{(Ay_s - Bx_s)}{\Delta} u_r + \frac{(-Ay_r - Bx_r)}{\Delta} u_s,$$

where Δ is the determinant of the inverse transformation,

$$\Delta = det \begin{bmatrix} x_r & y_r \\ x_s & y_s \end{bmatrix}.$$

With a little more fiddling, this can even be done in a conservative way (Viviand, [1978]). It is important to maintain the conservation form of differencing in problems with discontinuous solutions. To solve

$$u_t = f(u)_x + g(u)_y$$

under the more general coordinate transformation

$$r = r(x, y)$$
$$s = s(x, y)$$

we solve

$$u_t = \hat{f}(u)_r + g(u)_s$$

where

$$\hat{f} = \frac{(r_x f + r_y g)}{J}$$
$$\hat{g} = \frac{(s_x f + s_y g)}{J}.$$

In this transformation, J is the determinant

$$J = det \begin{bmatrix} r_x & r_y \\ s_x & s_y \end{bmatrix}.$$

We point out that it is sometimes not necessary to transform the difference equations for each grid. If the physical problem is invariant to translation and rotation, we can use the identical difference equations on each grid. However we might be solving for

different variables on a rotated grid. For example, in a gas dynamics problem we might solve for the velocity components in different coordinate directions on the different grids. The use of solution values from other grids must then be preceded by some interpolation. For scalar quantities, such as density, there is no additional complication. It can also happen that the difference scheme itself is invariant under rotation. Jameson [1974] has proposed a *rotated* difference scheme for transonic flow calculations. The rotation is built into the scheme to be able to properly difference the potential equations in the streamline and normal directions.

Finally, we note that it is sometimes beneficial to use different integrators in different regions of the solution. For example, in a shock problem one might use a first order integrator on a refined grid around a shock, and a higher order method elsewhere. Another approach could be to use a more expensive method such as the Glimm scheme only on refined grids, and a less expensive integrator on the coarse grid. One can even solve different equations on different grids. For example, it is possible to only add artificial viscosity on a fine grid (around a shock zone), or solve more difficult boundary layer equations only on a separate grid in the boundary layer. Several integration modules can be easily supplied by the user without any further changes to the mesh refinement program.

Error estimation and the subsequent regridding operation is the second major task of the mesh refinement algorithm. This is where most of the computational overhead of the method lies. Every several time steps, we estimate the error at all grid points and possibly create new fine grids or remove those no longer needed. If a new fine grid is created, its initial values are interpolated using the finest grids from the already existing grid structure. Nothing need be done to remove a grid that is no longer needed exept reclaim its storage space.

In the next section we will discuss the algorithm we use to estimate the error. In chapter 4 we will discuss our method of grid generation, which uses the results of the error estimation. Here we discuss how often this procedure should be done. In hyperbolic problems, one can estimate the speed of propagation and calculate how fast some phenomenon needing to be in a fine grid will move. If we add a buffer zone around the fine grid we can lengthen the time interval over which grids are appropriate, and thus lengthen the interval between the regridding operations. The larger this buffer zone the

less often we have to regrid, but the more work it is to integrate the extra points in the buffer zone.

An optimal size for this buffer zone can be found by minimizing the integration cost associated with it. Consider the scalar wave equation $u_t = au_x$ with one level of refinement. The buffer zone must have enough grid points so that in the number of time steps n between regridding, a wave doesn't escape from the fine grid. We know that a is the maximum wave speed. In n time steps of size k_c each, the wave travels $ank_c/h_c = n\lambda$ coarse grid points. We use λ for the Courant number $\lambda = ak/h$. The buffer zone must be at least this large on both sides of the refined grid. We determine n by minimizing the integration cost of the buffer zone plus the cost of estimating the error on the coarse grid. We wish to minimize

$$\min_n \quad \left(2n\lambda r^2 c_I T + c_E N \frac{T}{n}\right)$$

where

n = number of coarse time steps between regridding
c_E = cost /pt. of regridding
c_I = cost /pt. of integration
N = number of coarse grid points
r = refinement ratio
T = total number of coarse integration steps to be taken.

The minimum occurs for

$$n = \sqrt{\frac{c_E}{c_I}\frac{N}{2r^2\lambda}}.$$

Typical numbers of $\lambda \approx 1$, $r = 4$, $c_E/c_I \approx 3$, and 50 coarse grid points give an optimal regridding frequency of 2.3 time steps, or approximately every couple of coarse time steps.

If there are many levels of refinement, we apply this result at each level. The finest grids must be moved more often than the coarsest grids. This is easy to implement. We simply count how often the parent grid of the fine grid is integrated, and check it before integrating the fine grid itself. We call the same regridding procedure with a base level which is finer than the coarsest level. The base grids stay fixed, and finer subgrids will be created within the boundaries of the grids at the base level according to the proper nesting restrictions of §2.1. The potential problem here is that a refined grid might move off its base or not stay sufficiently far away from the base grid boundaries. If this happens, it is probably time to move the base grids as well.

The last major component of the mesh refinement algorithm concerns the grid communication routines. There are three tasks which fit under this heading. The first of these deals with boundaries. Refined grids have boundaries which are in the interior of the problem domain, and so they will need boundary values not supplied with the pde. These values can be calculated in three ways. First, if a boundary point is in the interior of a different fine grid, we can get the boundary value at the next time step from the interior integration of the intersecting grid. If there are no intersecting fine grids, the boundary values are calculated using values from underlying coarse grids. We use either the Coarse Mesh Approximation Method or interpolation to get the boundary values. These methods are fully described in §3.1. We point out here that the order of grid integration is important for implementing these boundary conditions. If a fine grid will get its boundary values at time $t + k_f$ from the coarse grid at time t, (by the Coarse Mesh Method), the fine grid should be integrated first, or else some coarse grid values must be saved. If the boundary values come from the coarse grid at time $t + k_c$, (by interpolation), then the coarser grid can be integrated first.

The second item of inter-grid communication is updating. If a fine grid is nested in a coarse grid, then when they are integrated to the same point in time the coarse grid values are updated by injecting the fine grid solution values onto the coarse grid points. If grid points do not match up at regular intervals, interpolation is used to find the value from the fine grid which replaces the coarse grid value. For explicit methods, it is only necessary to update as many coarse grid points at the perimeter of the fine grid as the number of points in the stencil for the coarse grid integrator. For implicit methods, it is necessary to update the entire coarse grid, and in fact, some sort of averaging might be desirable before injection onto the coarse grid. This is similar to the residual weighting used in multi-grid methods (Brandt, [1977]).

This updating of the coarser grids is what we call an active mesh refinement strategy as opposed to a passive one. It is necessary for several reasons. If the coarse grid is not updated, the solution can become so dispersed and dissipated that after a while it will look as if refinement is no longer necessary. Secondly, by updating the coarse grid and keeping a discontinuity sharp, the size of the zone needing refinement can be smaller. Finally, this prevents a train of bad values on the coarse grid from spreading

into the buffer zone and contaminating the values that will be used for the boundary approximation for the fine grid.

The last grid communication task is that of averaging. This only arises when two subgrids at the same level of refinement overlap. In general, the region of overlap is at most a few coarse mesh widths wide. Still, the question arises when updating the underlying coarse grid, which fine grid should inject the solution values. We have no clear answer to this yet. What we do now is average the fine grid values before injection. It is not necessary to replace either of the fine grid solutions with this averaged value however. The solution on either fine grid is as accurate as the other. Since they are only overlapping by $O(h)$ width, and they get the boundary values from each other, the solutions do not diverge from each other. However, if one is not careful about injecting onto the coarse grid, there can be a jump in the solution representation on the coarse grid. So far, this has only been important for graphical output.

The overall mesh refinement algorithm is presented in figure 2.4 in outline form. It can be written quite simply as a recursive procedure. Of course, in writing the mesh refinement program in Fortran, we have converted it to an iterative procedure.

```
Recursive Procedure Integrate (level)
    Repeat (h_f/h_c)^level times
        Regridding time? — error estimation for grids at level level and finer
        Step Δt_level on all grids at level (level)
        If (level + 1 exists)
        Then Begin
            Integrate (level + 1)
            Update (level, level + 1)
            End
    end.
level = 0 (* coarsest grid level *)
Integrate (level)
```

Figure 2.4 Coarse Grid Integration Cycle

2.3 Error Estimation

In this section we discuss the criteria on which grid refinement is based. It is not accidental that the section heading is *error estimation*; we base the refinement decision on estimates of the local truncation error using a procedure we describe later. It has been more common in the literature to base the refinement decision on other properties of the solution than its estimated error. Gropp [1980] used the gradient of the solution as his criteria in his simple feasibility study for two-dimensional mesh refinement. However, this is often not sufficient. A linear solution with a large gradient does not need refinement. Gradient checking will not catch the corners of ill-behaved solutions that do need refinement. Checking the gradient is useful only for a limited class of problems whose solution behavior is known in advance. Dwyer *et al* [1978] uses a combination of the gradient and curvature of the solution for their refinement criteria. They choose coordinate points ξ such that $d\xi \approx a|\frac{\partial T}{\partial S}|dS + b|\frac{\partial^2 T}{\partial S^2}|dS$, where S is the arc length along the other coordinate line $\eta = constant$, and T is the dependent variable (temperature in Dwyer's problem). Another choice is seen in the work of Winkler [1977], where his adaptive criteria is to choose ξ to make $\log^2 \frac{|\partial_\xi u_i|}{|u_i|}$ approximately equal at each grid point. All these methods work well under the assumptions of the problems they are designed for, but they are not generally applicable. We seek a more systematic approach, without assumptions on the direction of motion or number of regions needing refinement. Of course, if the user does have more information about the solution, this can be used to advantage.

We point out that the refinement criteria used in finite element calculations for non-hyperbolic equations does not seem to be as big a question as it is for finite difference methods. A natural error estimate is provided by the discrete bilinear form of the pde. This is used, for example, in solving elliptic problems (Babuška and Rheinboldt, [1979]), as well as parabolic problems (Dupont, [1982]). Davis and Flaherty [1982] use the fact that the error in a finite element approximation can be bounded by an interpolation error. They estimate the appropriate derivatives using interpolating polynomials and equidistribute this error bound for a nearly optimal mesh. Keith Miller *et al* [1981] move their finite elements by combining the residual minimization and the solution process itself. They use a finite element approximation of $u_t = L(u)$, where both the value and location of the node points change with time. The error they minimize is the L_2 norm

of the residual $R = v_t - L(v)$, where v is the solution from the approximating subspace. Miller's procedure has been justified theoretically by Dupont [1982].

The approach we use in estimating the error as a criterion for deciding where to refine the grid is to estimate the local truncation error. There are two reasons for this. First, we were motivated by the convergence results of (Gustafsson, [1975]) for initial boundary value problems for hyperbolic systems. Under some assumptions that are mostly about the Cauchy stability of the difference approximations and stability in the sense of Kreiss for the initial boundary value problem, Gustafsson shows that if

(*i*) the local truncation error $\approx kh^m d_\nu(t)$

(*ii*) the initial error of the approximation $\approx h^\beta e_\nu(\sigma\Delta t), \sigma = 0, 1, \ldots, s$

(*iii*) the error for the boundary approximation $\approx h^\beta f_\nu(t)$

where d, e, f are bounded functions, and if $\beta \geq m$, then the convergence rate is order m. Mesh refinement can control these errors by decreasing k and h in those regions where $|h^m d_\nu(t)|$ or $|h^\beta f_\nu(t)|$ are too large.

Gustafsson's results are for non-adaptive, uniform grid calculations. More recently, Oliger (Berger, Gropp and Oliger, [1982]) has a convergence result for adaptive mesh refinement under some stronger hypotheses than those of Gustafsson. He shows using a stronger definition of stability than Gustafsson's, that the global error for adaptive mesh refinement calculations is bounded by the local truncation error multiplied by some growth factors, assuming the interface equations are of the same order of accuracy as the interior equations. Experimentally, the expected rate of convergence is observed along with the expected decrease in constants when the local truncation error tolerance (that is, the refinement criteria) is reduced.

The method we use to estimate the error is based on Richardson extrapolation. For simplicity, let Q be a two-level explicit difference operator. If the solution is smooth enough, the local truncation error is

$$\begin{aligned} \|u(t+k) - Qu(t)\| &= k(k^{q_1}a(t) + h^{q_2}b(t)) + k\,O(k^{q_1+1} + h^{q_2+1}) \\ &= \tau + k\,O(k^{q_1+1} + h^{q_2+1}), \end{aligned} \tag{2.3.1}$$

where we denote the leading term by τ. If u is smooth enough, then if we take time two steps with the method Q, to leading order the error is 2τ,

$$\|u(t+2k) - Q^2u(t)\| = 2\tau + k\,O(k^{q_1+1} + h^{q_2+1}).$$

Let Q_{2h} be the same difference method as Q but based on mesh widths of $2h$ and $2k$. Also, assume the order of accuracy in time and space are equal, $q_1 = q_2$. Then

$$\begin{aligned} \|u(t+2k) - Q_{2h}u(t)\| &= (2k)\left((2k)^q a(t) + (2h)^q b(t)\right) + O\left(h^{q+2}\right) \\ &= 2^{q+1}\tau + O\left(h^{q+2}\right). \end{aligned}$$

Since $\|u(t+2k) - Q^2u(t)\| \approx 2\tau$, by forming the difference

$$\frac{Q^2u(t) - Q_{2h}u(t)}{2^{q+1} - 2} = \tau + O\left(h^{q+2}\right), \tag{2.3.2}$$

we get an estimate of the local truncation error at time t. In words, we take one *giant* step based on mesh widths of $2h$ and $2k$ using the solution at time t and compare it with the solution obtained by taking two regular integration steps to obtain the pointwise error estimate. This is illustrated schematically in figure 2.5.

This procedure has several points to recommend it. First, it is not necessary to know the exact form of the truncation error to apply it. The functions $a(t)$ and $b(t)$ in (2.3.1), which involve higher derivatives of the solution, need not be known explicitly. Especially for systems of equations in several variables, it can be quite difficult to compute the exact form of the error. Not only is our estimator independent of the pde, the error estimation procedure is independent of the difference method. This is important if mesh refinement is going to be generally applicable to a wide variety of problems without an

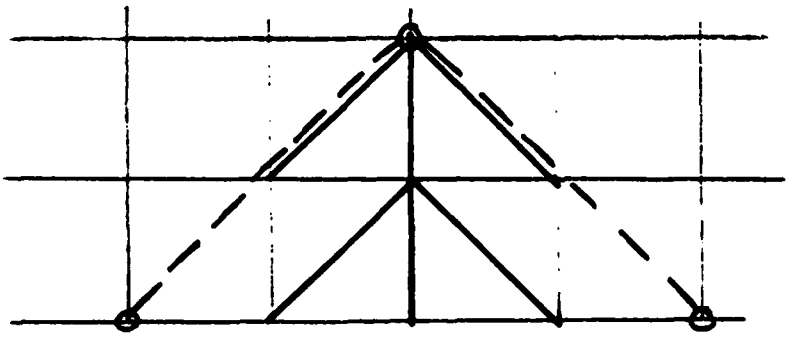


Figure 2.5 Richardson Error Estimation Procedure

inordinate amount of programming. The exact same user-supplied method of integration can be used for the error estimation. The restriction of this procedure, that the accuracy in time and space be the same, is not a severe one. Many popular finite difference methods fall into this category, for example, second order methods like Lax-Wendroff or MacCormack's method, and Leap Frog, and first order methods such as usptream differencing. For methods where the accuracy in space and time is not the same, a more expensive variant of this procedure is possible. For example, one could estimate the spatial and temporal error separately, by first keeping k constant and taking a step based on $2h$ differences, then keep h constant and take a step with time step $2k$. Other variations are possible too.

The overhead of this method falls into the usual space - time tradeoff. With careful programming, the two steps with the regular difference operator Q_h can be part of the regular time integration of the pde, in which case extra storage must be used to save the solution $u(t)$, compute $Q_{2h}(u(t))$, and compare it with $Q_h^2(u(t))$. The principal overhead is the one integration step in time plus two temporary arrays of storage. If the overhead is increased to 2 integration steps, only 1 solution array is necessary. Since this procedure is done for one grid at a time, the extra solution array is only the size of a grid, not the size of the total discretization.

For a better idea of the accuracy of the Richardson estimates, we examine the error in the error estimator. We compute the difference between the estimate of the local truncation error and the actual error as a function of frequency. We derive an expression for the minimum number of points per wavelength below which the grid will be refined, valid only for a fixed point in time. Such expressions can be computed for the accuracy of resolving certain wavelengths of the pde itself (Kreiss and Oliger, [1972]; Trefethen, [1982]).

We consider the scalar wave equation

$$u_t = u_x \tag{2.3.3}$$

with an initial function of a single wave number $u(x,0) = e^{\imath\xi x}$. We approximate (2.3.3) using the Lax Wendroff difference scheme

$$v_j^{n+1} = v_j^n + \frac{\lambda}{2}(v_{j+1}^n - v_{j-1}^n) + \frac{\lambda^2}{2}(v_{j+1}^n - 2v_j^n + v_{j-1}^n), \tag{2.3.4}$$

where the mesh ratio $\lambda = k/h$. The symbol of (2.3.4), denoted p_ξ, is

$$p_\xi = 1 + \imath\lambda \sin(\xi h) - 2\lambda^2 \sin^2(\xi h/2).$$

Two steps of Lax-Wendroff are equivalent to multiplication of the initial conditions by p_ξ^2.

Let z_ξ denote the symbol of Lax Wendroff applied on a $2h - 2k$ grid,

$$z_\xi = 1 + i\lambda \sin(2\xi h) - 2\lambda^2 \sin^2(\xi h).$$

Since Lax Wendroff is second order, by (2.3.2) the error estimate is

$$e_{est} = \frac{p_\xi^2 - z_\xi}{6}.$$

A little algebra yields the expression

$$\frac{p_\xi^2 - z}{6} = \frac{8i\lambda(1-\lambda^2)\sin^3\beta\cos\beta + 4\lambda^2(\lambda^2-1)\sin^4\beta}{6}, \tag{2.3.5}$$

where $\beta = \xi h/2$. The first (imaginary) term represents the estimated phase error, and the second the estimated error in the amplitude.

We compare this using the first two terms of the exact truncation error for Lax Wendroff on (2.3.3). The local truncation error for Lax Wendroff can be written

$$\begin{aligned} u_j(t+k) - Qu_j(t) &= \frac{k}{6}(k^2-h^2)u_{j_{xxx}}(t) + \frac{k^2}{24}(k^2-h^2)u_{j_{xxxx}}(t)\dots \\ &= (\frac{8i\lambda(1-\lambda^2)}{6}\beta^3 - \frac{4\lambda^2(1-\lambda^2)}{6}\beta^4)e^{i\xi x} + O(\beta^5). \end{aligned} \tag{2.3.6}$$

This shows the surprising fact that the estimate of the truncation error is second order accurate even though in general it is only first order. However, since we do not use the error estimate to explicitly correct the solution, this is of little consequence.

Suppose we require the local error estimate

$$\left|\frac{p_\xi^2 - z_\xi}{6}\right| < \epsilon,$$

for some prescribed tolerance ϵ. We compute the number of points per wavelength below which the accuracy criterion is not met, and the grid is refined. Use $\lambda = .8$ for a typical value of the mesh ratio. Then we get the following table.

ϵ	N_{est}	N_{exact}
.1	4.07	4.97
.05	5.62	6.24
.01	10.30	10.62
.005	13.12	13.37
.001	22.70	22.85
.0005	28.66	28.78

N_{est} is computed using (2.3.5). N_{exact} is computed using the first two terms of (2.3.6). We see that for smooth solutions our error estimator is extremely accurate.

There are several points that should be mentioned before concluding this section. We have not made any attempt here to describe a procedure for estimating the error at the boundary of the domain. In addition, the error estimation procedure uses a stencil twice as wide as the numerical method itself. For numerical methods with more than a three point stencil, there is therefore a significant region near the boundary that can't be estimated by the above procedure. It is sometimes possible to use the numerical boundary conditions on a $2h - 2k$ grid, or a special one-sided scheme to estimate at the boundary. This is discussed in detail bt Bolstad [1982]. Another possibility is to use fictitious points outside the boundary, and apply the interior procedure at all points. This is discussed further in (Berger, Gropp, Oliger, [1982]).

For non-smooth solutions, the analysis above breaks down. We no longer have an accurate error estimate, and in fact the local truncation error is not well defined. However, the Richardson estimates still provide a good criterion for refinement, since, near a singularity the estimate will probably be large. The magnitude of the estimate will be off, but the location will still be correct. For piecewise constant initial conditions, the Richardson algorithm gives an estimate proportional to the jump in the solution. We mention that with the procedure above, in the neighborhood of a shock the maximum number of levels of refinement will be used. If this is not desirable then some other criterion, such as an estimate of the l_1 norm of the error (which converges at a shock as $h \rightarrow 0$) should be used instead.

Finally, we mention the possibility of using other criteria along with the local truncation error as a mesh refinement trigger. For example, one could refine the grid to retain monotonicity of the solution. In (van Leer, [1977]) there is a full discussion of monotonicity properties of difference schemes.

Chapter 3: INTERFACE CONDITIONS

In this chapter, we present the difference approximations we use at the interfaces of the fine and coarse grids. More in keeping with the viewpoint of treating each grid as an independent entity, we restate the problem as finding boundary conditions for the fine grids. The boundaries of the fine grids will in general be interior to the problem domain, and the boundary values will come from the coarse grids. In §3.1 we describe the most common interface conditions we use in both 1 and 2 space dimensions. In §3.2 we present a new stability proof for one of the interface conditions in one space dimension. This is the only stability result for refineed meshes with different time steps. In §3.3 we discuss conservation properties of the mesh refinement algorithm, and return to boundary conditions by presenting a procedure to derive conservative interface conditions for any difference scheme. We use it to derive some conservative boundary conditions for use with the Lax-Wendroff difference scheme, and discuss their accuracy and stability.

We do not discuss here or elsewhere in this thesis the boundary conditions needed by the coarse grids at the boundary of the computational domain. The coarse grid will have some boundary conditions supplied with the pde, supplemented by numerical boundary conditions supplied by a user. This is a difficult but standard problem for which there are several major references (Engquist and Majda, [1977]; Gustafsson, Kreiss and Sundström, [1972]; Trefethen, [1982]). Anyone who solves a pde confronts this problem with or without mesh refinement. Our aim is to produce a method to automatically handle the extra complexity introduced by using mesh refinement.

3.1 Boundary Condition Algorithms

The first boundary condition we consider is the Coarse Mesh Approximation Method (CMAM). This was used by Ciment [1971]. We want to solve the scalar wave equation

$$u_t = u_x$$

in one space dimension. Let the coarse grid be on the left of the interface, with approximate solution at the coarse grid points labelled $v_0, v_{-1}, v_{-2}, \ldots$, as you move left. Let a fine grid which has been refined by a factor of n be on the right, with approximate

solution values $u_0, u_1, u_2, \ldots$ as you move right. We have $\Delta x_f = \Delta x_c/n, \Delta t_f = \Delta t_c/n, \lambda$ constant on both grids. The notation is the same as in figure 3.1. Let Q be an explicit finite difference approximation applied on the coarse grid,

$$v_j^{m+1} = Qv_j^m.$$

The CMAM formula determines the boundary values for $u_0^i, 0 < i \leq n$ using the same difference approximation Q with a smaller time step, and with the values needed to the right of v_0 supplied by u_n, u_{2n}, etc. Note that if the fine mesh finite difference scheme has a stencil with more than three points, additional boundary values will be needed. These can be obtained using the same stencil but shifted right by one or more fine mesh points. This will require spatial interpolation in the coarse grid.

If Q is a two time level formula, for example Lax-Wendroff, the boundary values will be formally as accurate as the interior, coarse grid calculations. If Q is more than two time levels, for example Leap Frog, then the accuracy depends on whether or not there are previous values at the interface at the right time. Since Q is a Cauchy stable scheme for the mesh ratio $\lambda = \Delta t_c/\Delta x_c$ of the coarse grid, this interface condition is Cauchy stable for the smaller ratio $\frac{k}{n}\frac{\Delta t_f}{\Delta x_f}$, $1 \leq k \leq n$.

A disadvantage of the CMAM is that it requires the fine and coarse grids to meet exactly at a grid point. In general in two dimensions this won't happen. One could interpolate in space using an interpolation formula of as high order as the finite difference method, and then apply the coarse grid method. However we prefer the simpler Interpolation Boundary Condition algorithm of using linear interpolation in time along with bilinear interpolation in space for a two dimensional computation. Of course higher order interpolations in time are possible with the additional storage overhead of the extra levels in time. We point out that for some first order methods in time, for example Lax-Friedrichs or first-order upwind differencing, interpolation and the CMAM are identical. However, interpolation has advantages over the CMAM with non-linear problems when artificial viscosity is added. In addition, the interpolation boundary conditions make it possible to use an implicit integration method on the fine and coarse grids.

Finally, for completeness we repeat the description of a boundary condition we use in two or more dimensions. (It is first mentioned in §2.2.) The boundary values for a fine grid might be determined by interpolation using values from the interior of an intersecting fine grid that contains the boundary point. There are several considerations here that

we have not yet explored. For example, if the intersecting grids are in a shock zone, care could be taken not to interpolate across the shock. On the other hand, it might be beneficial to explicitly add artificial viscosity to the interpolation procedure.

3.2 Stability of an Interface Condition

In this section we will derive an expression and prove stability for one of the boundary conditions discussed in §3.1. We consider the Lax-Wendroff difference scheme applied to $u_t = u_x$, with linear interpolation at the interface of the fine and coarse grid. The stability definition we will use is due to (Gustafsson, Kreiss, Sundström, [1972]). We will briefly state their results that we will use here, and assume the reader is already familiar with their paper.

The definition of stability they use for the initial boundary value problem (IBVP) appears as definition 3.3 in their paper. They consider the quarter plane problem

$$u_t = Au_x + Bu + F(x,t), \tag{3.2.1}$$

for $0 \leq x \leq \infty$, where A, B are constant n - by - n matrices, u and $F \in \Re^n$.

Definition: The approximation Q is stable when applied to homogeneous initial data if there are constants $K_0 > 0, \alpha_0 \geq 0$ such that for all $\alpha > \alpha_0$ and all F and g we have the estimate

$$\left(\frac{\alpha-\alpha_0}{1+\alpha k}\right)\sum_{\nu=0}^{l-1}\|e^{-\alpha t}v_\nu\|_t^2 + \left(\frac{\alpha-\alpha_0}{1+\alpha k}\right)^2\|e^{-\alpha t}v\|_{x,t}^2 \leq K_0^2\left[\left(\frac{\alpha-\alpha_0}{1+\alpha k}\right)\sum_{\nu=0}^{l-1}\|e^{-\alpha(t+k)}g_\nu\|_t^2 + \|e^{-\alpha(t+k)}F\|_{x,t}^2\right] \tag{3.2.2}$$

for all sufficiently small k.

In this definition $t = mk$, g is the boundary data, and F is a forcing function. The norms in x and t are the usual l_2 norms

$$\|v_\nu\|_t^2 = k\sum_{m=0}^{\infty}|v_\nu^m|^2$$

$$\|v\|_{x,t}^2 = hk\sum_{m=0}^{\infty}\sum_{\nu=0}^{\infty}|v_\nu^m|^2$$

We make several remarks about this definition before we state their main result. The estimate (3.2.2) involves norms of the solution in space and time. It is not known whether (3.2.2) implies there are l_2 estimates in space at fixed times. In adaptive mesh refinement calculations, the solution at a given time t_0 provides the initial conditions for the next strip of calculations using a different grid structure for $t > t_0$. One could probably use the l_2

estimate in space to prove convergence of adaptive mesh refinement involving regridding. However we will be proving stability of non-adaptive mesh refinement according to stability definition (3.2.2).

The main result in Gustafsson *et al* is the following:

Theorem: Let Q be a Cauchy stable difference approximation to the quarter plane initial boundary value problem, and let Q be either strictly dissipative or non-dissipative. A necessary and sufficient condition for Q to be stable according to definition (3.2.2) is that the associated homogeneous resolvent equations have no eigenvalues or generalized eigenvalues with $|z| \geq 1$.

Their theorem tells us to look for eigensolutions which grow geometrically in time. The resolvent equations are the equations that result after a Laplace transform of the difference equations in time. For an s time-level approximation that is applied at all gridpoints up to but not including the boundary point, the homogeneous resolvent equations look like

$$\left(Q_{-1} - \sum_{\sigma=0}^{s} z^{-\sigma-1} Q_\sigma\right) \hat{w}_\nu = 0, \quad \nu = 1, 2, \ldots \tag{3.2.3}$$

with boundary equation

$$\hat{w}_0 - \sum_{\sigma=1}^{s} z^{-\sigma-1} S_\sigma \hat{w}_1 = 0.$$

Equation (3.2.3) is a constant coefficient difference equation with general solution in l_2 of the form

$$\hat{w}_\nu = \sum_{|\kappa_j| \leq 1} P(\nu, j) \kappa_j^\nu(z).$$

The degree of the polynomial P is one less than the multiplicity of the corresponding root κ_ν. Also, the sum includes roots $|\kappa_j| = 1$ only if they are roots which approach the boundary of the unit circle from the inside as $|z| \to 1$ from outside the unit circle. Thus we know (in this case) the form of the eigensolutions is

$$v_\nu^m = z^m \sum_j P(\nu, j) \kappa_j^\nu(z).$$

These normal modes satisfy both the interior difference equations and the boundary conditions. If there are no such eigensolutions or generalized eigensolutions with $|z| \geq 1$ then the approximation is stable.

The last point we make before deriving the stability condition for mesh refinement concerns the so-called *folding trick* (Ciment, [1972]). Consider a scalar problem with an interface at $x = 0$, with one difference scheme on the left and a possibly different difference scheme on the right, and some interface equations. The stability of the combined scheme can be determined by folding the left side along the line $x = 0$, thus transforming the problem into an equivalent $2 - by - 2$ system of equations for $x \geq 0$, with boundary conditions determined by the interface equations. Then we can apply the initial boundary value theory. For example, to approximate $u_t = au_x$ we can approximate the equivalent system

$$\begin{pmatrix} u_1 \\ u_2 \end{pmatrix}_t = \begin{pmatrix} a & 0 \\ 0 & -a \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}_x,$$

where $u_1(x) = u(x)$ and $u_2(x) = u(-x)$ for $x > 0$. It is unnecessary to actually go through the *folding* before applying the IBVP theory, especially since it tends to confuse the notation. We point out that by using the initial boundary value theory and definition (3.2.2), the solution at the interface will receive special weight that does not appear in definitions of Cauchy stability. Trefethen [1982] has shown that if one applies the folding trick to the Cauchy problem for $u_t = u_x$, and applies Leap Frog at all grid points including the interface point, the result has a generalized eigenvalue.

We can now derive the stability condition for $u_t = u_x$ approximated by Lax-Wendroff on both the coarse and fine grids. Let the fine grid be refined by an arbitrary integral factor n. An equal refinement factor in time and space is not an essential part of this derivation, but for simplicity we will assume it. Several facts about Lax-Wendroff will be important in this proof.

The situation we model is illustrated in figure 3.1.

In figure 3.1 we have drawn the intermediate timesteps on the fine grid, but in fact we can view the difference scheme on the right as being one application of Q^n, where Q is the Lax-Wendroff operator with mesh ratio λ. The GKS theory says to look for modes which grow geometrically in time, and which repeat themselves at every timestep. Let this factor in time be z^n. Let our difference operator Q be a three point centered scheme. By lemma 6.2 in Gustafsson, *et al*, there are as many roots κ_j, $|\kappa_j| < 1$, of the characteristic equation as there are points to the right of center in the stencil for the difference operator. Correspondingly, there are as many roots with $|\kappa_j| > 1$ as there are points to the left. In particualar, to get the characteristic equation for Lax-Wendroff

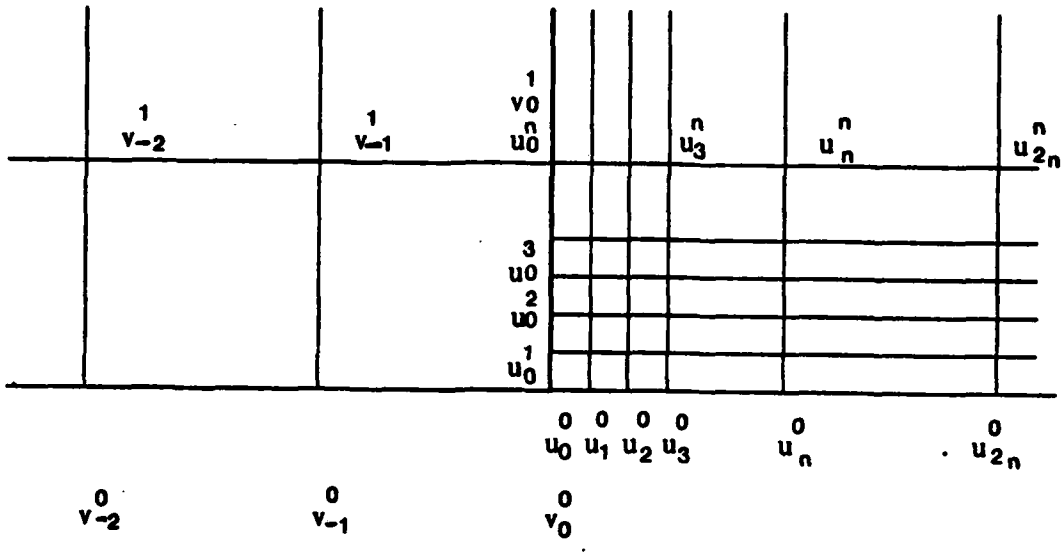


Figure 3.1

$$v_\nu^{m+1} = v_\nu^m + \frac{\lambda}{2}(v_{\nu+1}^m - v_{\nu-1}^m) + \frac{\lambda^2}{2}(v_{\nu+1}^m - 2v_\nu^m + v_{\nu-1}^m)$$

we substitute *constant* $z^m\kappa^\nu$ for v_ν^m, to obtain

$$z = 1 + \frac{\lambda}{2}(\kappa - \kappa^{-1}) + \frac{\lambda^2}{2}(\kappa - 2 + \kappa^{-1})$$

with two roots

$$\kappa_\pm = \frac{z - 1 + \lambda^2 \pm \sqrt{(z-1)^2 + \lambda^2(2z-1)}}{\lambda^2 + \lambda}. \tag{3.2.4}$$

The root κ_- is inside the unit circle, κ_+ is outside.

On the left of the interface for the coarse grid we therefore look for modes of the form $v_\nu^m = (z^n)^m\tau^\nu$, where for simplicity we will normalize the constant to 1. We use τ for the root of the resolvent equation for $z < 0$. Since we are numbering the coarse gridpoints with negative indices, for the solution v to be in l_2 we want the root τ for which $|\tau| > 1$.

On the right of the interface, n applications of Q has a stencil n points to the right of center. By the lemma above, there must be n roots of the difference equation inside the unit circle. Let $\omega_j, j = 0, 1, \ldots, n-1$ be the n roots of unity; $z\omega_j$ are then the n roots of z^n. For every value $z\omega_j$ there is a corresponding κ_j by (3.2.4). Each such mode κ_j satisfies the requirement that after one global repetition of the difference equations,

or n steps on the fine grid, the solution in space is multiplied by z^n. There are n such modes κ_j, and since each $z_j = z\omega_j$ with modulus > 1 determines only one root inside the unit circle, the κ_j are distinct. The general solution represented on the fine grid must then be of the form

$$u_\nu^m = \sum_{j=0}^{n-1} \rho_j \kappa_j^\nu (z\omega_j)^m,$$

where the ρ_j are constants. The picture is the following.

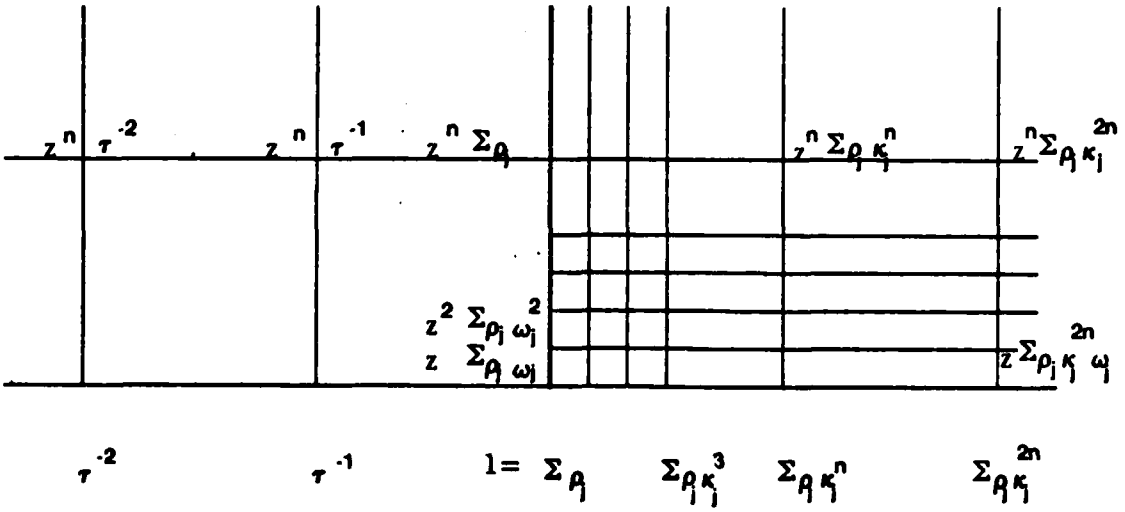


Figure 3.2

We point out that the solution on either side of the interface must take the form of the normal modes at all gridpoints all the way up to the boundary. On the left, we have a three point scheme which can be applied at all gridpoints but v_0, where we get to specify one boundary condition. Similarly, at each fine grid timestep we can apply Lax Wendroff at each point $u_j, j \geq 1$, and specify n conditions for the $u_0^i, i = 1, \ldots, n$.

Now we examine the interface conditions that these modes must satisfy. We require first that $v_0^1 = u_0^n$, or

$$\sum_j \rho_j = 1. \tag{3.2.5}$$

To compute v_0^1 we use a coarse mesh approximation, and use the gridpoint u_n^0 for v_1^0. We know that if three points are geometrically related in space at time t, then the value of z that multiplies this mode in time is fully determined. Similarly, if we know two points at time t and the z factor at time $t + \Delta t$, the third point at time t is also already determined.

This means that the point u_n^0 must fit into the normal mode of the coarse grid as the point v_1^0. This gives us the relation $u_n^0 = v_1^0$, or

$$\tau = \sum_j \rho_j \kappa_j^n. \tag{3.2.6}$$

For the remaining $n-1$ gridpoints u_0^k at the interface we will use linear interpolation,

$$u_0^k = v_0^0 + \frac{k}{n}(v_0^1 - v_0^0), \quad 1 \le k \le n-1$$

giving us the equations

$$1 + \frac{k}{n}(z^n - 1) = z^k \sum_j \rho_j \omega_j^k, \quad k = 1, \ldots, n-1. \tag{3.2.7}$$

Equations (3.2.5) and (3.2.7) together give n conditions that the constants $\rho_j, j = 0, \ldots, n-1$ must satisfy. We can write this as the linear system $W\underline{\rho} = \underline{b}$,

$$\begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ \omega_0 & \omega_1 & \omega_2 & \ldots & \omega_{n-1} \\ \omega_0^2 & \omega_1^2 & \omega_2^2 & \ldots & \omega_{n-1}^2 \\ \vdots & \vdots & \vdots & & \vdots \\ \omega_0^{n-1} & \omega_1^{n-1} & \omega_2^{n-1} & \ldots & \omega_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} \rho_0 \\ \rho_1 \\ \rho_2 \\ \vdots \\ \rho_{n-1} \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{1}{z} + \frac{1}{n}\frac{(z^n-1)}{z} \\ \frac{1}{z^2} + \frac{2}{n}\frac{(z^n-1)}{z^2} \\ \vdots \\ \frac{1}{z^{n-1}} + \frac{n-1}{n}\frac{(z^n-1)}{z^{n-1}} \end{pmatrix}.$$

Since the matrix W on the left hand side is a Van der Monde matrix, and the roots of unity are distinct, we can solve uniquely for the ρ_j. Furthermore, if we normalize the matrix W, and define $F = \frac{1}{\sqrt{n}}W$, then F is unitary, $F^*F = I$.

We then solve for $\underline{\rho} = F^*\underline{b}$ to get

$$\rho_j = \frac{1}{n}\left[\sum_{k=0}^{n-1}(\frac{\overline{\omega}_j}{z})^k\right] + \frac{(z^n-1)}{n^2}\left[\sum_{k=0}^{n-1} k(\frac{\overline{\omega}_j}{z})^k\right]. \tag{3.2.8}$$

The two terms in (3.2.8) can be summed explicitly. In particular we use the relation

$$\sum_{k=0}^{n-1} kr^k = \frac{r}{(1-r)^2}\left[r^n(n-1) - nr^{n-1} + 1\right].$$

Summing and simplifying finally gives us an expression for the constants

$$\rho_j = \frac{(z^n-1)^2}{(z-\overline{\omega}_j)^2}\frac{\overline{\omega}_j}{n^2 z^{n-1}}.$$

There is still one equation from the interface conditions, (3.2.6), that we haven't used. Substituting the expression for ρ_j into (3.2.6) gives us the *root condition*

$$\tau = \sum_{j=0}^{n-1} \left(\frac{z^n - 1}{z - \varpi_j} \right)^2 \frac{\varpi_j}{n^2 z^{n-1}} \kappa_j^n. \tag{3.2.9}$$

Let us summarize at this point. If there are eigenvalues or generalized eigenvalues that satisfy both the difference equations and interface conditions, they must satisfy (3.2.9). If (3.2.9) has a solution with $|z| \geq 1$ then the difference equations are unstable. We will prove the following

Theorem: Using Lax-Wendroff with interpolation interface conditions for the fine grid, mesh refinement by any integer is stable.

Proof: To prove this, we must show the root condition (3.2.9) can't hold for $|z| \geq 1$. The main idea of the proof is to apply the maximum principle (see for example Ahlfors, [1966]) to the right-hand side of the root condition (3.2.9), call it $f(z)$. We will show that for $|z| > 1$ there are no branch points, and $f(z)$ is regular at ∞. Thus the maximum modulus of $f(z)$ occurs for $|z| = 1$. We know from (3.2.4) that τ, the left hand side of (3.2.8), satisfies $|\tau| \geq 1$. The last part of the proof consists of showing that the modulus of $f(z)$, $|f(z)| \leq 1 - \delta, \delta > 0$, for $|z| = 1$, and so the root condition cannot hold.

First we check for branch points in the region $|z| > 1$. From (3.2.4), this occurs for

$$(z-1)^2 + \lambda^2(2z-1) = 0.$$

This is a quadratic in z with roots

$$z_\pm = (1 - \lambda^2) \pm i\lambda\sqrt{1-\lambda^2}.$$

The modulus of the roots is

$$\begin{aligned} |z|^2 &= (1-\lambda^2)^2 + \lambda^2(1-\lambda^2) \\ &= 1 - \lambda^2 \\ &\leq 1 \end{aligned},$$

since for Cauchy stability for Lax-Wendroff, $0 \leq \lambda \leq 1$. So for $|z| > 1$ there are no branch points.

Next we examine the expression

$$f(z) = \sum_{j=0}^{n-1} \left(\frac{z^n - 1}{z - \varpi_j} \right)^2 \frac{\varpi_j}{n^2 z^{n-1}} \kappa_j^n$$

for large $|z|$, by checking the behavior of the κ_j as $|z| \to \infty$. We have

$$\kappa = \frac{z - 1 + \lambda^2 - \sqrt{(z-1)^2 + \lambda^2(2z-1)}}{\lambda^2 + \lambda}$$

where we take the branch of the square root where $\sqrt{z^2} = z$. For large $|z|$ we write

$$\kappa = \frac{z + \lambda^2 - 1 - z\sqrt{1 - \frac{2}{z} + \frac{1}{z^2} + \frac{2\lambda^2}{z} - \frac{\lambda^2}{z^2} + O(z^{-3})}}{\lambda^2 + \lambda}.$$

The binomial expansion for

$$(1-\epsilon)^{1/2} = 1 - \frac{\epsilon}{2} - \frac{\epsilon^2}{8} - \cdots$$

also gives us the correct branch. Putting it all together we get, for each $\kappa_j = \kappa_j(z_j)$, where $z_j = z\omega_j$,

$$\kappa_j = \frac{-\lambda(1-\lambda)}{2z_j} + O\left(\frac{1}{z_j^2}\right).$$

Thus we get

$$f(z) = \sum_j \frac{\overline{\omega}_j[(-\lambda)(1-\lambda)]^n}{2^n n^2}\frac{1}{z} + O\left(\frac{1}{z^2}\right),$$

which clearly goes to zero for large $|z|$.

We digress and point out that the root τ has the asymptotic behavior in z of

$$\tau = \frac{2z^n - \frac{3}{2}(1-\lambda^2)}{\lambda^2 + \lambda} + O\left(\frac{1}{z^n}\right),$$

and so the root condition can't hold for $|z| \to \infty$.

Since $f(z)$ is regular at ∞ and has no branch points, the maximum modulus principle, which states that *if f is analytic in the interior and continuous up to the boundary of a region R, then the maximum modulus* $|f(z)|$ *occurs on the boundary.* We will show that the maximum modulus of f is less than, and bounded away from 1.

It is easy to show, as is done in lemma 6.1 of Gustafsson *et al*, that the roots $\kappa_\pm$ of Lax-Wendroff for $u_t = u_x$ satisfy

$$\begin{aligned} |\kappa_-| &\le 1 - \delta \qquad |z| \ge 1 \\ |\kappa_+| &> 1 \qquad |z| \ge 1,\ z \ne 1. \\ \kappa_+ &= 1 \qquad z = 1 \end{aligned}$$

Therefore the root τ of (3.2.9) satisfies $|\tau| \ge 1$ for $|z| \ge 1$, and all the κ_j satisfy $|\kappa_j| \le 1 - \delta$. We therefore have

$$|f(z)| \le \frac{(1-\delta)}{n^2}\sum_j \left|\frac{z^n - 1}{z - \overline{\omega}_j}\right|^2. \tag{3.2.10}$$

It remains to show that

$$\sum_{j=0}^{n-1} |\frac{z^n - 1}{z - \omega_j}|^2 = n^2 \tag{3.2.11}$$

when $|z| = 1$. Since this expression is equivalent to (3.2.11), we will drop the annoying complex conjugate bars and instead prove the identity

$$\sum_{j=0}^{n-1} |\frac{z^n - 1}{z - \omega_j}|^2 = n^2 \tag{3.2.11}$$

Examine the expression $\sum_j |\frac{z^n-1}{z-\omega_j}|^2$, which is the square of the discrete l_2 norm of a vector $\underset{\sim}{e}$ with j^{th} component $e_j = \frac{z^n-1}{z-\omega_j}$. By Parseval's theorem, the norm of a vector is the same as the norm of its finite Fourier transform, $||e||_2 = ||\hat{\underset{\sim}{e}}||_2$, which we now compute. We form

$$\hat{\underset{\sim}{e}} = \frac{1}{\sqrt{n}} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ \omega_0 & \omega_1 & \omega_2 & \dots & \omega_{n-1} \\ \omega_0^2 & \omega_1^2 & \omega_2^2 & \dots & \omega_{n-1}^2 \\ \vdots & \vdots & \vdots & & \vdots \\ \omega_0^{n-1} & \omega_1^{n-1} & \omega_2^{n-1} & \dots & \omega_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} e_0 \\ e_1 \\ e_2 \\ \vdots \\ e_{n-1} \end{pmatrix},$$

do the matrix multiplication, and get for the k^{th} component of the vector $\hat{\underset{\sim}{e}}$,

$$\hat{e}_k = \frac{1}{\sqrt{n}} \sum_j \frac{z^n - 1}{z - \omega_j} \omega_j^k.$$

By explicitly doing the division

$$\frac{z^n - \omega_j^n}{z - \omega_j} = \sum_{l=0}^{n-1} z^l \omega^{n-l},$$

we obtain

$$\hat{e}_k = \frac{1}{\sqrt{n}} \sum_j \sum_l z^{n-1-l} \omega_j^{l+k}.$$

We switch the order of summation and consider

$$\sum_j \omega_j^{l+k} = \sum_j (\omega_0^{l+k})^j,$$

where ω_0 is the first primitive root of unity. For $l + k \neq 0$ or n, this sum is

$$\frac{1 - (\omega_0^{l+k})^n}{1 - (\omega_0^{l+k})} = 0.$$

For $l + k = 0$ or n, the sum is equal to n. For every $k > 0$ there is only one l between 0 and $n-1$ satisfying $l = n-k$, so $\hat{e}_k = \frac{n}{\sqrt{n}} z^{k-1}$, for $k = 1, \ldots, n-1$, and $\hat{e}_0 = \frac{n}{\sqrt{n}} z^{n-1}$. Altogether, we have

$$\underset{\sim}{\hat{e}} = \frac{n}{\sqrt{n}} \begin{pmatrix} z^{n-1} \\ 1 \\ z \\ z^2 \\ \vdots \\ z^{n-2} \end{pmatrix}.$$

So for $|z| = 1$, we compute

$$\|\underset{\sim}{\hat{e}}\|_2^2 = n^2.$$

Therefore in (3.2.10), $|f(z)| \leq (1 - \delta)$, whereas $|\tau| \geq 1$. Thus the root condition cannot hold for $|z| \geq 1$, and so our approximation is stable. ∎

3.3 Conservation

We begin this section with a general discussion of conservation properties of both the differential and difference equations. We briefly review why conservation is important in computing discontinuous solutions. We then discuss in particular the conservation properties of mesh refinement. Our discussion is for mesh refinement in one space dimension, although some of our results carry over to higher dimensions. It is in higher dimensions that conservation becomes important, and there are other questions that remain to be addressed in these cases. We present a procedure for deriving conservative boundary conditions in one space dimension. We use it to derive several interface conditions for use with Lax-Wendroff at internal mesh boundaries, and discuss their stability and accuracy.

The general hyperbolic conservation law system in one dimension is written

$$u_t = f(u)_x \tag{3.3.1}$$

with initial conditions

$$u(x,0) = u_0(x).$$

If we define the quantity $S = \int_{x_1}^{x_2} u\,dx$ we find

$$dS/dt = f(x_2) - f(x_1),$$

or the change in the quantity S is due only to the flux at the boundaries. If there is no flux across the boundary, or the domain is periodic in which case $f(u(x_2)) = f(u(x_1))$, then the quantity S is conserved.

In some calculations, it is important that the numerical method as well as the pde exactly conserve some solution quantities. This is true for example in the equations of gas dynamics with discontinuous solutions, where the total mass should be conserved, or for another example, in long-term weather forecasting models (Arakawa, [1966]). Of course for smooth solutions, if the difference method is stable and sufficiently accurate, quantities which the physical problem conserves will be approximately conserved numerically. Also, if the quantity to be conserved is positive, for example if $\int u^2$ is to be conserved numerically, then stability of the difference approximation is assured.

An explicit finite difference approximation to (3.3.1) is said to be in *conservation form* if it can be written

$$\begin{aligned} v_j^{m+1} &= Q(v_{j-p}^m, \ldots, v_{j+q+1}^m) \\ &= v_j^m + \lambda \Delta_+ g(v_{j-p}^m, \ldots, v_{j+q}^m), \end{aligned} \tag{3.3.2}$$

where $g(v, v, \ldots, v) = f(v)$. In (3.3.2) we use $\lambda = \Delta t/\Delta x$, and Δ_+ is the forward difference operator, $\Delta_+ f(x) = f(x+h) - f(x)$. If a difference approximation can be written in the form (3.3.2) where g is Lipschitz continuous, it is consistent with (3.3.1). The one-step Lax-Wendroff difference scheme can be written in conservation form by choosing

$$g(v_{j-1}, v_j) = \frac{f(v_{j-1}) + f(v_j)}{2} + \frac{\lambda}{2} A\left(\frac{v_{j-1} + v_j}{2}\right)(f(v_j) - f(v_{j-1}))$$

where the Jacobian $A = \partial f/\partial u$.

It is important to write the equation in conservation form for several reasons. In problems with a discontinuous solution, classical derivatives of the solution no longer exist, and a weak solution is defined instead. (It is well known [Lax, (1973)] that this is not enough to determine a unique weak solution.) If u is a weak solution of (3.3.1) then

$$\int\int (\varphi_t u - \varphi_x f)\, dx\, dt + \int u_0(x)\varphi(x,0)\, dx = 0 \qquad (3.3.3)$$

holds for all smooth test functions $\varphi(x,t)$ with compact support. If u is discontinuous, the local truncation error is no longer well-defined. In some sense, (3.3.3) provides a way to define consistency in these cases. Lax and Wendroff [1960] proved that if v is an approximation in conservation form that converges boundedly almost everywhere as $h, k \to 0$ to a function $u(x,t)$, then u is a weak solution of the conservation law (3.3.1). A stronger result has been obtained for a scalar conservation law by Harten, Hyman and Lax [1976] in 1 space dimension, and by Crandall and Majda [1977] in n dimensions, that monotone finite difference schemes in conservation form converge to the correct, entropy satisfying weak solution of the differential equation. Furthermore, in problems with a shock or contact discontinuity in the solution, the shock speed s is given by the Rankine-Hugoniot conditions,

$$s[u] = [f],$$

where the brackets denote the jump in the quantity at the discontinuity. This condition follows directly from (3.3.3) (Lax, [1973]). It guarantees that the numerical shock speed is correct. It is known (Noh, [1976]) that schemes which are consistent for smooth solutions but not in conservation form may sometimes compute the wrong shock speed.

In this section we will discuss conservation as it applies to adaptive mesh refinement. There are several places in the mesh refinement algorithm that conservation can break

down: at the interfaces between the fine and coarse grids, and during regridding - both creating and removing fine grids. In one dimension, if we are approximating a discontinuous solution there should always be a fine grid in the region of the discontinuity, and so the conservation properties of regridding are not critical. In two dimensions however, shocks can cross subgrid boundaries, and subgrids containing a shock can change orientation during regridding. In these cases, exact conservation is important. We will consider in turn these three places in the algorithm where conservation may be lacking. We will show how to derive a class of conservative interface conditions that give weak solutions to the differential equation if they converge.

To develop the procedure, we first consider conservative boundary conditions for the case of mesh refinement in space only. For ease of presentation, we consider the case of refinement by a factor of 2. The same procedure generalized immediately to larger refinement ratios. Assume the interface between the fine and coarse grids is located at $x = 0$, where we use the notation in figure 3.3.

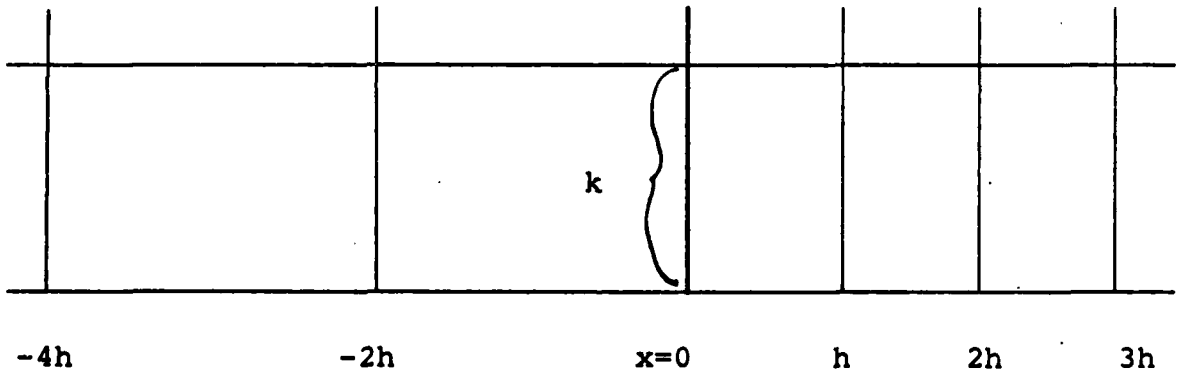


Figure 3.3

We will derive the boundary conditions assuming a scheme in conservation form with a three point stencil is used on either side of the interface. The scheme is

$$\frac{v(x,t+k)-v(x,t)}{k} = \begin{cases} \frac{g(x,x+h)-g(x-h,x)}{h}, & x \geq h \\ \frac{g(x,x+2h)-g(x-2h,x)}{2h}, & x \leq -2h \\ Q(-2h,0,h), & x = 0. \end{cases} \qquad (3.3.4)$$

Here, we use $g(x,y)$ to mean $g(u(x),u(y))$. The goal is to determine the unknown function Q for the interface equation for $v(0,t)$ in such a way as to maintain conservation.

Following Lax and Wendroff, we multiply (3.3.4) by a smooth test function $\varphi(x,t)$ and by the mesh widths in time and space and sum over all gridpoints x and $t \geq 0$. The outline is the following. Summation by parts will put the differences on the test function φ instead of the numerical flux function g. As $h, k \to 0$, the differences should converge to derivatives of φ and the sums to integrals in space and time. There is some flexibility as to which discrete integral approximation is used, although any integration formula will give a consistent boundary approximation. For the summation in space, for example, we can consider

$$\int_{-\infty}^{\infty} v(x,t)\,dx \approx \sum_{x \leq -2h} v(x,t)2h + \sum_{x \geq h} v(x,t)h + \frac{3h}{2}v(0,t). \tag{3.3.5}$$

We interpret the sum to be over only the grid points in each half plane considered. This approximation comes from a trapezoid rule approximation to the integral. Another way to look at it is that the grid function $v(x_j,t)$ is extended by linear interpolation for values of x which are not grid points, in which case the integration is exact. We could also consider

$$\int_{-\infty}^{\infty} v(x,t)\,dx \approx \sum_{x \leq -2h} v(x,t)2h + \sum_{x \geq 0} v(x,t)h, \tag{3.3.6}$$

which comes from a one-sided approximation to the integral, or a one-sided extension of the grid function as constant over the interval to the right. These different extensions of the grid function will give different conservative boundary conditions.

First consider the summation in time of the left hand side of the difference scheme (3.3.4) when multiplied by the test function $\varphi(x,t)$,

$$\sum_{t=0}^{\infty}\left[\frac{v(x,t+k)-v(x,t)}{k}\right]\varphi(x,t)k = \sum_{t \geq k} v(x,t)\left[\frac{\varphi(x,t-k)-\varphi(x,t)}{k}\right]k - v(x,0)\varphi(x,0),$$

where we sum over values of t which are an integral multiple of k. Since φ is a smooth test function the backward differences in time converge to a derivative. We also assume that as $k \to 0$, the approximate solutions $v(x,t)$ converge boundedly pointwise almost everywhere, so the sum will converge to an integral in time. For this side of the equation, it remains to do the summation in space, which will present no problem.

Next consider the summation in space of the right hand side of (3.3.4) using the one-sided integral approximation (3.3.6),

$$\sum_{x\le -2h}\left[\frac{g(x,x+2h)-g(x-2h,x)}{2h}\right]\varphi(x,t)2h+\sum_{x\ge h}\left[\frac{g(x,x+h)-g(x-h,x)}{h}\right]\varphi(x,t)h$$
$$+hQ(-2h,0,h)\varphi(0,t).$$

Applying summation by parts to both sums gives

$$\sum_{x\le -2h} g(x,x+2h)\left[\frac{\varphi(x,t)-\varphi(x+2h,t)}{2h}\right]2h \quad + \quad g(-2h,0)\varphi(-2h,t)$$
$$+\sum_{x\ge h} g(x,x+h)\left[\frac{\varphi(x,t)-\varphi(x+h,t)}{h}\right]h \quad - \quad g(0,h)\varphi(h,t)$$
$$+hQ(-2h,0,h)\varphi(0,t).$$

The two sums together already form an $O(h)$ approximation to the integral in space. We must have that the three terms remaining cancel up to terms of order h, or

$$hQ(-2h,0,h)\varphi(0,t)+g(-2h,0)\varphi(-2h,t)-g(0,h)\varphi(h,t)=O(h).$$

Expanding the test function $\varphi(x,t)$ around $x=0$ gives the expression

$$[hQ+g(-2h,0)-g(0,h)]\,\varphi(0,t)+O(h)=O(h),$$

and so we will determine Q by requiring

$$hQ+g(0,h)-g(-2h,0)=0. \tag{3.3.7}$$

Before actually determining Q we finish the argument that Q will in fact give a weak solution if the approximations v converge. So far we have only looked at the left hand side of (3.3.4) being summed in t, and the right hand side summed in space. Doing both to both sides of the equation (3.3.4) gives

$$\sum_{x\le -2h}\sum_{t\ge k} v(x,t)\left[\frac{\varphi(x,t-k)-\varphi(x,t)}{k}\right]k\,2h+$$
$$\sum_{x\ge 0}\sum_{t\ge k} v(x,t)\left[\frac{\varphi(x,t-k)-\varphi(x,t)}{k}\right]k\,h-$$
$$\sum_{x\le -2h} v(x,0)\varphi(x,0)2h-\sum_{x\ge h} v(x,0)\varphi(x,0)h=$$
$$\sum_{t\ge 0}\sum_{x\le -2h} g(x,x+2h)\left[\frac{\varphi(x,t)-\varphi(x+2h,t)}{2h}\right]k\,2h+$$
$$\sum_{t\ge 0}\sum_{x\ge h} g(x,x+h)\left[\frac{\varphi(x,t)-\varphi(x+h,t)}{h}\right]k\,h+O(h).$$

As in the original theorem, we take limits as $h, k \to 0$. The sums become integrals, the differences in φ become derivatives, and the flux function $g(x, x+h)$ becomes $g(x,x)$ or $f(x)$, by the original consistency requirement for the numerical flux function g. We get

$$\int\int v(x,t)\varphi_t(x,t) - f(v)\varphi_x(x,t)\,dx\,dt = -\int v(x,0)\varphi(x,0)\,dx.$$

Note that this is independent of the integration formula used.

For a particular example we use the definition of the numerical flux function g for Lax-Wendroff, noting that the mesh ratio λ of the fine grid must be doubled for the coarse grid. Using (3.3.7) we get

$$hQ = \frac{f(0)+f(h)}{2} + \frac{\lambda}{2}A(\frac{h}{2})(f(h0 - f(0)) - \left(\frac{f(-2h)+f(0)}{2} + \lambda A(-h)(f(0)-f(-2h))\right).$$

This gives us the interface approximation

$$v(0,t+k) = v(0,t) + \frac{\lambda}{2}(f(h)-f(-2h)) + \frac{\lambda^2}{2}\left[A(\frac{h}{2})(f(h)-f(0)) - 2A(-h)(f(0)-f(-2h))\right].$$

This is not a very accurate approximation, however. For smooth solutions the local truncation error is only $O(h)$. However, since the scheme is only applied at one point we can still get convergence if the scheme is GKS-stable, since the method applied everywhere else is at least $O(kh)$ (Gustafsson, [1976]). A more accurate approximation results from the trapezoid approximation (3.3.5) to the integral. The summation in time is identical, but for the summation in space we get

$$\sum_{x\le -2h}\left[\frac{g(x,x+2h)-g(x-2h,x)}{2h}\right]\varphi(x,t)2h + \sum_{x\ge h}\left[\frac{g(x,x+h)-g(x-h,x)}{h}\right]\varphi(x,t)h + \frac{3h}{2}Q(-2h,0,h)\varphi(0,t).$$

Repeating the process of summation by parts gives sums which again are an $O(h)$ approximation to the integral and some terms which are $O(1)$. Again this determines the conservation boundary condition Q for this integral approximation, by requiring the $O(1)$ terms to exactly cancel. In this case we get for the interface approximation

$$\frac{3h}{2}Q(-2h,0,h) = g(0,h) - g(-2h,0)$$

which gives the interface approximation

$$v(0,t+k) = v(0,t) + \lambda\left(\frac{f(h)-f(-2h)}{3}\right) + \frac{\lambda^2}{3}\left[A(\frac{h}{2})(f(h)-f(0)) - 2A(-h)(f(0)-f(-2h))\right]. \tag{3.3.8}$$

This is a more accurate approximation, which we will show later when we discuss exact discrete conservation. The stability of this approximation will also be considered later.

The class of boundary schemes we just derived are for the case of mesh refinement in space. For mesh refinement in time and space, very little changes in the derivation. Again consider the case of mesh refinement by a factor of 2, as in figure 3.4.

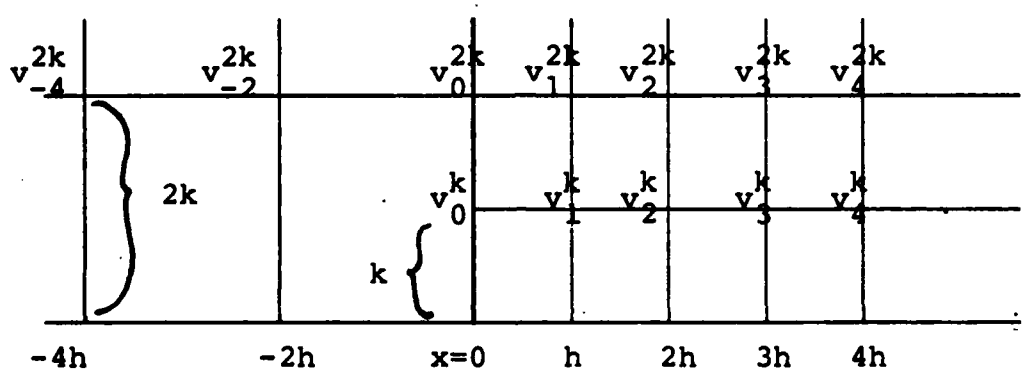


Figure 3.4

Suppose for notational convenience we write the trapezoid approximation in space

$$\int v(x,t)\,dx \approx \sum_{x\le -2h} v(x,t)2h + \sum_{x\ge 0} \alpha_j v(x,t)h$$

where the weights α_j are defined by $\alpha_0 = 3/2$, $\alpha_j = 1$ for $j \ge 1$. For the spatial sums on both the left and right of the interface it is clear how to do the sums in time for a natural approximation to the integral in time. We form

$$\sum_{t\ge 0}\sum_{x\le -2h} v(x,t)2h2k + \sum_{x\ge 0}\sum_{t\ge 0} \alpha_j v(x,t)hk$$

and again apply summation by parts to derive the interface conditions. However, we postpone the discussion of this interface condition until a different derivation is presented later when considering discrete conservation.

It is clear that given any integration rule and numerical flux function, conservative boundary conditions can be defined so that the approximation converges to a weak solution. We now discuss exact discrete conservation as an end in itself. Our discussion will center on Lax-Wendroff and adaptive mesh refinement, but what we do here can be done for other schemes. In this case, the aim is to exactly conserve a discrete form of $\int u\,dx$ independent of the grid structure of the computation.

We consider the approximation of (3.3.1) by Lax-Wendroff,

$$v_j^{m+1} = v_j^m + \frac{\lambda}{2}(f_{j+1}^m - f_{j-1}^m) + \frac{\lambda^2}{2}[A_{j+\frac{1}{2}}^m(f_{j+1}^m - f_j^m) - A_{j-\frac{1}{2}}^m(f_j^m - f_{j-1}^m)].$$

For computational efficiency it is common to use $A(v_{j+\frac{1}{2}}) \approx \frac{1}{2}(A(v_{j+1}) + A(v_j))$. We change notation here and write

$$v_j^{m+1} = v_j^m + \frac{\lambda}{2}(f_{j+1}^m - f_{j-1}^m) + \frac{\lambda^2}{2}(g_{j+\frac{1}{2}}^m - g_{j-\frac{1}{2}}^m)$$

where we are now defining $g_{j+\frac{1}{2}} = A_{j+\frac{1}{2}}(f_{j+1} - f_j)$. If we are solving the problem on a uniform grid with mesh width h, with grid points numbered from $0, 1, \ldots, n+1$, it is easy to see that

$$h\sum_{j=1}^{n} v_j^1 = h\sum_{j=1}^{n} v_j^0 + \frac{k}{2}(f_{n+1}^0 + f_n^0 - f_1^0 - f_0^0) + \frac{k\lambda}{2}(g_{n+\frac{1}{2}}^0 - g_{\frac{1}{2}}^0), \tag{3.3.9}$$

or the change in $h\sum v_j$ is due to the numerical fluxes at the boundary. This follows directly from the definition for any scheme that can be written in conservation form.

We first consider the change in $S(v)$ that occurs when a new fine grid is created. We look at one coarse mesh width where this occurs, and denote the left and right grid points v_0 and v_r. Let the interval have mesh width h and let the fine grid be refined by a factor of r, so the new fine grid points can be labelled $v_1, \ldots, v_{r-1}$. The subscripts f and c will denote fine and coarse grid quantities respectively. We will use the trapezoid rule

$$S_c = h\frac{(v_0 + v_r)}{2}$$

$$S_f = \frac{h}{r}\sum_{j=1}^{r}\frac{(v_{j-1} + v_j)}{2}$$

to approximate the integral $\int v\,dx$.

Suppose the new fine grid values will be gotten by linear interpolation,

$$v_j = (1 - \frac{j}{r})v_0 + \frac{j}{r}v_r, \quad 1 \le j \le r-1.$$

We compute

$$S_f = \frac{h}{r}\sum_{j=1}^{r-1} v_j + \frac{h}{2r}(v_0 + v_r).$$

The first sum gives

$$\begin{aligned}\frac{h}{r}\sum_{j=1}^{r-1} v_j &= \frac{h}{r}\sum_{j=1}^{r-1}(1-\frac{j}{r})v_0 + \frac{h}{r}\sum_{j=1}^{r-1}\frac{j}{r}v_r \\ &= \frac{h}{2}(v_0+v_r) - \frac{h}{2r}(v_0+v_r),\end{aligned}$$

and so the total quantity

$$S_f = \frac{h}{2}(v_0 + v_r) = S_c.$$

We say that the trapezoid rule approximation to the integral $\int u\,dx$ and linear interpolation are *compatible* operations. For more accuracy in the computation we can generalize this. Let S be a p panel integration rule with weights w_i,

$$S(f) = \sum_{i=0}^{p} w_i u(x+ih) = \int_x^{x+ph} u(x)\,dx + O(h^{q+1})$$

We have the

Proposition: Let S_c be the approximation to the integral $\int u\,dx$ on the coarse grid,

$$S_c = \int u\,dx + O(h^{q+1}).$$

Then if you use interpolation of order q to set the new fine grid values, it follows that

$$S_c(\underset{\sim}{v}_c) = S_f(I_c^f \underset{\sim}{v}_c),$$

or conservation is maintained.

We borrow from multigrid notation (Brandt, [1976]) in writing the approximate solution on the fine grid which is interpolated from the coarse grid as $\underset{\sim}{v}_f = I_c^f \underset{\sim}{v}_c$. This proposition follows from the fact that the integration rule S is based on interpolating the function values by a polynomial of degree q and integrating exactly. But since the fine grid values are themselves determined by interpolation by a polynomial of degree q, both the coarse and fine grid integral approximations give the same value.

This would be an irrelevant observation if the difference scheme itself did not conserve the more accurate integral approximation. However, we quote the following lemma from (Mock and Lax, [1978]) which we use to show that arbitrarily accurate integral

approximations are conserved by a difference scheme in conservation form, except at the boundaries. They show

Lemma: Let f be any C^∞ function on R_+ with bounded support. Given any positive integer ν, there exists a quadrature formula accurate of order ν of the form

$$\int_0^\infty f(x)\,dx = h\sum_0^\infty w_j f(jh) + O(h^\nu),$$

where the weights w_j depend on ν, but $w_j = 1$ for $j \geq \nu$.

Since the weights are constants equal to 1 in the interior, a scheme in conservation form will conserve this accurate approximation up to the boundary region.

In any case, we thus have that with this restriction of compatible integration/interpolation rules, newly created fine grids can be made to be conservative and as accurate as desired. A little more must be done to remedy the lack of conservation when deleting fine grids. Again we use the trapezoid rule to get

$$S_c = \frac{h}{2}(v_0 + v_r),$$

and

$$S_f = \frac{h}{2r}\sum_{j=1}^{r}(v_{j-1} + v_j).$$

The discrete gridpoints on the fine grid can be interpolated by an r^{th} degree polynomial on the interval $[x_0, x_r]$, and so we can interpret $v(x)$ for x not a gridpoint to mean $v(x) = p_r(x)$. Also let r be even for easier notation. Then by Taylor expansions we have

$$\begin{aligned} S_f &= \frac{h}{2r}\sum_{j=1}^{\frac{r}{2}} 2v_0 + (2j-1)hv_{0x} + \frac{h}{2r}\sum_{j=\frac{r}{2}+1}^{r} 2v_r - (2j-1)hv_{rx} + O(h^2) \\ &= \frac{h}{2}(v_0 + v_r) + \frac{h^2 r}{8}(v_{0x} - v_{rx}) + O(h^3) \end{aligned}$$

If v is smooth enough,

$$S_f = S_c + \frac{h^3 r}{8} v_{\frac{r}{2}xx} + O(h^4),$$

and so the lack of conservation is rather small when removing a fine grid.

In the most important cases however u will not be smooth. Some kind of procedure, analogous to the residual weighting and injection used in multigrid algorithms is called for here. We calculate the difference between S_c and S_f, and the coarse grid points are perturbed enough to make the sums equal. For smooth solutions, we saw above that the

perturbation using a 2^{nd} order quadrature rule is the same size as the truncation error for a 2^{nd} order method. We illustrate by deriving the formula for the perturbation using the trapezoid rule formulas on the fine and coarse grids. We distribute the difference in S due to the first half of the interval to the left coarse grid point, and from the right half of the interval to the right coarse grid point. If r is even, there is a point midway between the two coarse grid points which should be shared. For the case of mesh refinement by 2 this looks like

$$v'_r = \frac{1}{4}v_{r-1} + \frac{1}{2}v_r + \frac{1}{4}v_{r+1},$$

whereas for mesh refinement by 3, we use equal weights

$$v'_r = \frac{1}{3}v_{r-1} + \frac{1}{3}v_r + \frac{1}{3}v_{r+1}.$$

Again, for mesh refinement by 4 this is

$$v'_r = \frac{1}{8}v_{r-2} + \frac{1}{4}v_{r-1} + \frac{1}{4}v_r + \frac{1}{4}v_{r+1} + \frac{1}{8}v_{r+2}.$$

Thus this procedure is basically an averaging, or smoothing process. This is reasonable since the higher frequencies which were resolved on the fine grid should be smoothed before injecting them onto the coarse grid. Presumably, if the high frequencies were important, the fine grid would still be necessary to resolve them, and it would not be removed. We remark that the formulas used to perturb a coarse grid point on the boundary of a fine grid are a little different. They are the equivalent one-sided update formulas. For example, for refinement by 4 at the left boundary point we would use

$$v'_0 = \frac{5}{8}v_0 + \frac{2}{8}v_1 + \frac{1}{8}v_2.$$

The weights for the points to the left have been added to the weight for the boundary point instead. This same redistribution can also be done for higher order integral approximations to maintain conservation.

This idea of distributing the difference among the remaining grid points in order to conserve the chosen quantities is found in a slightly different context in Sasaki [1976] and Isaacson [1977]. In these papers, the solution to the finite difference scheme is modified after each step so that certain functionals of the solution take prescribed values. These functionals are the discrete conservation relations. The modification of smallest norm to the value predicted by the finite difference scheme is chosen. The two papers differ

in the way of solving for the smallest norm modification – Lagrange multipliers versus linearizing and setting the first variation to zero. Isaacson has used his procedure to conserve total mass and energy in a one layer atmospheric model on the sphere using overlapping grids in different coordinate systems. A procedure like this could be done for mesh refinement in several dimensions to conserve when there are overlapping refined grids.

We interject several remarks here. It seems likely that the discrete conservation we are talking about in creating and removing fine grids will play a role in the proof of convergence of the approximations to weak solutions in *adaptive* mesh refinement computations. When integrating in time over each strip of a computation where the grids stay fixed, summation by parts will give boundary terms which should exactly cancel if the creating and deleting of grids has been done conservatively. Finally, we admit that conservation in two space dimensions has additional problems that we leave for future research. This is due in part to rotated grids not meeting at grid point boundaries, complicated by the overlapping fine grid structure.

The last problem of conservation for mesh refinement computations comes from the difference equations used at the interface of the fine and coarse grid. We have already considered this in the context of weak solutions and conservation in the limit as $h, k \to 0$. In that case, we multiplied by a smooth test function φ and integrated the $O(h)$ approximations to the integral. If we take the smooth test function φ to be constant in the neighborhood of the interface, we will get the conservative boundary conditions that will give exact conservation at each discrete time step. Instead of deriving the boundary conditions for space and time mesh refinement in that context however, we will do it from the point of view of exact discrete conservation. But we state that they are equivalent, and that the boundary conditions derived here do give weak solutions if the approximate solutions converge.

We first derive the expression for the change in the discrete sum $S(v)$ between timesteps. We will derive the expression for grids which are refined by a factor of 2; it is clear how to do it for larger factors. We will discuss the accuracy of the conservative boundary conditions also for refinement by 2 in space and time, but will only prove stability for the analogous conservative boundary condition for refinement in space only.

The grid notation we use is indicated in figure 3.5.

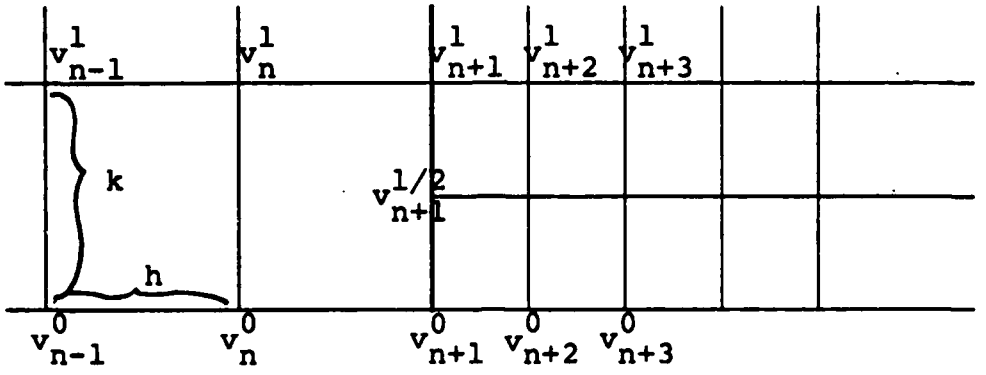


Figure 3.5

We will only look at the one interface between the grids, and assume the other end of each grid is also handled appropriately for conservation. The conditions we derive are for Lax-Wendroff. We change notation slightly and define $S_c = \sum_{j=1}^{n} v_j$, omitting the h.

From (3.3.9) we know that for the coarse grid,

$$S_c^1 = S_c^0 + \frac{\lambda}{2}(f_{n+1}^0 + f_n^0) + \frac{\lambda^2}{2} g_{n+\frac{1}{2}}^0. \tag{3.3.10}$$

Defining $S_f = \sum_{j=n+2}^{N_{max}} u_n$ gives

$$S_f^1 = S_f^0 - \frac{\lambda}{2}[f_{n+1}^0 + f_{n+2}^0 + f_{n+1}^{1/2} + f_{n+2}^{1/2}] - \frac{\lambda^2}{2}[g_{n+\frac{3}{2}}^0 + g_{n+\frac{3}{2}}^{1/2}]. \tag{3.3.11}$$

We define the total sum that we want to conserve on both the fine and coarse grids by using the trapezoid approximation,

$$I = hS_c + \frac{h}{2}S_f + \frac{3}{4}hv_{n+1}.$$

Putting (3.3.10) and (3.3.11) together, we can get $I^1 = I^0$ if we define the boundary equations

$$v_{n+1}^{1/2} = v_{n+1}^0 + \frac{\lambda}{3/2}\left[\frac{f_{n+1}^0 + f_{n+2}^0}{2} - \frac{(f_n^0 + f_{n+1}^0)}{2}\right] + \frac{\lambda^2}{3/2}\left[\frac{g_{n+\frac{3}{2}}^0}{2} - \frac{g_{n+\frac{1}{2}}^0}{4}\right], \tag{3.3.12}$$

and

$$v_{n+1}^1 = v_{n+1}^{1/2} + \frac{\lambda}{3/2}\left[\frac{f_{n+1}^{1/2} + f_{n+2}^{1/2}}{2} - \frac{(f_n^0 + f_{n+1}^0)}{2}\right] + \frac{\lambda^2}{3/2}\left[\frac{g_{n+3/2}^{1/2}}{2} - \frac{3}{4}g_{n+1/2}^0\right]. \tag{3.3.13}$$

These conditions have the interesting stencil

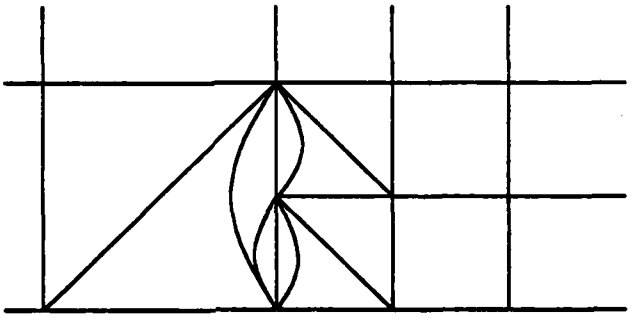


Figure 3.6

Intuitively, on each side of the interface the boundary point gets the flux it should from each grid.

We show now that for smooth solutions equations (3.3.12) and (3.3.13) are first order accurate in space and time. We use k and h as the mesh spacings for the fine grid. For the first half step we write

$$\begin{aligned} v_{n+1}^{1/2} &= v_{n+1}^0 + k\left[\frac{f_{n+2}^0 - f_n^0}{3h}\right] \\ &\quad + \frac{k^2}{2}\left[\frac{1/2(A_{n+2} + A_{n+1})(f_{n+2} - f_{n+1})}{\frac{3}{2}h^2} - \frac{1/2(A_n + A_{n+1})(f_{n+1} - f_n)}{3h^2}\right] \\ &= u_{n+1}^0 + k[f_x + O(h)] \\ &\quad + \frac{k^2}{2}\left[\frac{(A_{n+3/2} + O(h^2))(f_{xn+3/2} + O(h^2))}{\frac{3}{2}h}\right. \\ &\quad \left. - \frac{(A_{n+1/2} + O(h^2))(f_{xn+1/2} + O(h^2))}{\frac{3}{2}h}\right]. \end{aligned} \tag{3.3.14}$$

We see that the last term in (3.3.14) is

$$= \frac{k^2}{2}\left(\frac{A_{n+\frac{3}{2}} f_{xn+\frac{3}{2}} - A_{n+1/2} f_{xn+\frac{1}{2}}}{\frac{3}{2}h} + O(h)\right)$$
$$= \frac{k^2}{2}\left(\frac{\partial(A_{n+1} f_{xn+1})}{\partial x} + O(h)\right).$$

Altogther we have

$$u_{n+1}^{1/2} = u_{n+1}^0 + k f_x + \frac{k^2}{2}\frac{\partial(A f_x)}{\partial x} + O(kh).$$

Since $(A f_x)_x = (f_u u_t)_x = (f_t)_x = (f_x)_t = u_{tt}$, we have overall

$$u_{n+1}^{1/2} = u_{n+1}^0 + k u_t + \frac{k^2}{2} u_{tt} + O(kh),$$

and so the approximation is first order accurate in space and second order accurate in time.

However, for the rest of the interface approximations, the order of accuracy in time drops as well, due to the asymmetry of the stencil in time. The interesting thing here is how the asymmetry in time gives terms which contribute to the approximation of both the first and second derivatives. We first examine the terms in the interface approximation (3.3.13)

$$\frac{k}{3h}\left[f_{n+1}^{1/2} + f_{n+2}^{1/2} - (f_n^0 + f_{n+1}^0)\right]$$

which we rewrite as

$$\frac{k}{3h}\left[f_{n+2}^{1/2} - f_n^0 + f_{n+1}^{1/2} - f_{n+1}^0\right]$$

Taylor series expansions yield at the later time $t = k/2$,

$$\frac{k}{3h}\left[f_{n+2}^{1/2} - f_n^{1/2} + k f_{t_n}^{1/2} + O(k^2) + f_{n+1}^{1/2} - f_{n+1}^{1/2} + k f_{t_{n+1}}^{1/2}\right].$$

Again we use an uncentered first derivative approximation $f_{n+2} - f_n = 3h f_{xn+1} + O(h^2)$, and regroup terms to get

$$k f_{xn+1}^{1/2} - \frac{kh}{2} f_{xxn+1}^{1/2} + \frac{k^2}{3h}(f_{t_n}^{1/2} + f_{t_{n+1}}^{1/2}) + O(\lambda k^2). \qquad (3.3.15)$$

Looking now at the last term in (3.3.13) we have

$$\frac{\lambda^2}{3/2}\left[\frac{g_{n+3/2}^{1/2}}{2} - \frac{3}{4} g_{n+1/2}^0\right] =$$
$$\frac{k^2}{3h^2} g_{n+\frac{1}{2}}^{1/2} - \frac{k^2}{2h^2}(g_{n+\frac{1}{2}}^{1/2} - k g_{t_{n+\frac{1}{2}}}^{1/2}).$$

Replacing g by its definition,

$$g_{n+\frac{3}{2}} = hA_{n+\frac{3}{2}} f_{x\,n+\frac{3}{2}} + O(h^3)$$
$$g_{n+\frac{1}{2}} = 2hA_{n+\frac{1}{2}} f_{x\,n+\frac{1}{2}} + O(h^3)$$

the last term in (3.3.13) becomes

$$\frac{k^2}{3h} A_{n+3/2} f_{x\,n+3/2}^{1/2} - \frac{k^2}{h} A_{n+1/2} f_{x\,n+1/2}^{1/2} - \frac{k^2}{2h^2} g_{t\,n+1/2}^{1/2}.$$

We now combine the $g_{n+1/2}$ term with the t-derivative terms left over from the uncentered f_x approximation in the first term,

$$= \frac{k^2}{3h}\left(f_{t\,n}^{1/2} + f_{t\,n+1}^{1/2} - 3A_{n+\frac{1}{2}} f_{x\,n+\frac{1}{2}}^{1/2}\right)$$
$$= \frac{k^2}{3h}\left(2f_{t\,n+\frac{1}{2}}^{1/2} - 3A_{n+\frac{1}{2}} f_{x\,n+\frac{1}{2}}^{1/2}\right).$$

Since $f_t = f_u u_t = A f_x$, the entire expression above is

$$= \frac{k^2}{3h}\left(-A_{n+\frac{1}{2}} f_{x\,n+\frac{1}{2}}^{1/2}\right).$$

Putting it all together we have

$$u_{n+1}^1 = u_{n+1}^{1/2} + k f_{x\,n+1}^{1/2} - \frac{kh}{2} f_{xx\,n+1}^{1/2}$$
$$+ \frac{k^2}{3h}(A_{n+\frac{3}{2}} f_{x\,n+\frac{3}{2}} - A_{n+\frac{1}{2}} f_{x\,n+\frac{1}{2}}) + O\left(\frac{k^3}{h}\right)$$
$$= u_{n+1}^{1/2} + k f_{x\,n+1}^{1/2} + \frac{k^2}{2}(A_{n+1} f_{x\,n+1})_x + O(kh + \lambda k^2).$$

We again use the relation $(Af_x)_x = u_{tt}$ and get that (3.3.13) is a first order accurate approximation in space and time.

It is quite difficult to determine the stability of (3.3.12) and (3.3.13), although experimentally they appear to be stable. What we show here now is the GKS–stability of the equivalent conservative boundary approximation for the case of mesh refinement in space only,

$$u_{n+1}^1 = u_{n+1}^0 + \frac{\lambda}{3/2}(f_{n+2}^0 - f_n^0) + \frac{\lambda^2}{3/2}(2g_{n+\frac{3}{2}}^0 - g_{n+\frac{1}{2}}^0).$$

where we use $\lambda = \Delta t_c / \Delta x_c$. (This is the same conservative interface approximation we derived earlier in equation (3.3.8) when considering weak solutions for mesh refinement in space only.) We apply this approximation to the usual $u_t = u_x$, using Lax-Wendroff on both the fine and coarse grids. We use the notation in figure 3.7.

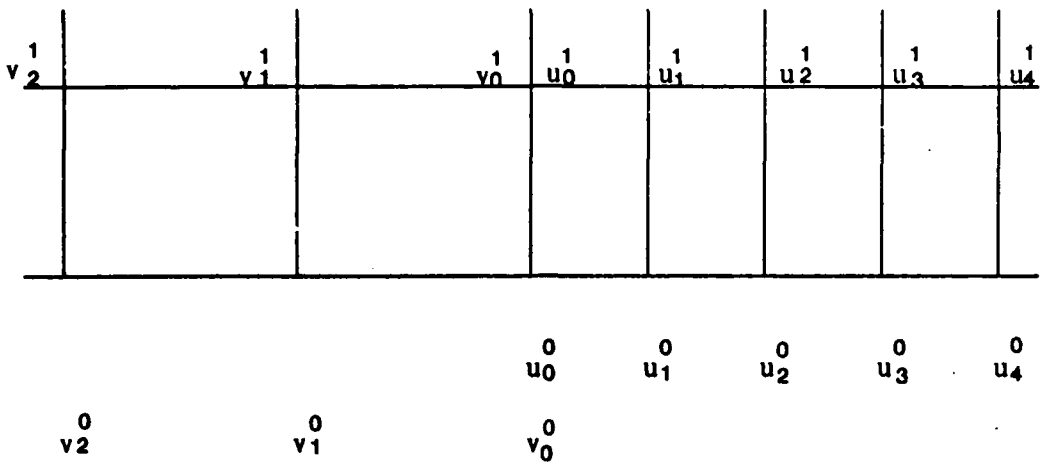


Figure 3.7

The interior difference approximations are

$$u_j^{m+1} = u_j^m + \lambda(u_{j+1} - u_{j-1}) + 2\lambda^2(u_{j+1} - 2u_j + u_{j-1})$$
$$v_j^{m+1} = v_j^m + \frac{\lambda}{2}(v_{j-1} - v_{j+1}) + \frac{\lambda^2}{2}(v_{j+1} - 2v_j + v_{j-1}).$$

The boundary conditions in this notation for these equations are

$$v_0^{m+1} = u_0^{m+1}$$
$$u_0^{m+1} = u_0^m + \frac{\lambda}{3/2}[u_1 - v_1] + \frac{\lambda^2}{3/2}[2(u_1 - u_0) - (v_0 - v_1)].$$

Again we look for normal modes which in this case are geometric in space and time,

$$u_j^m = \rho_1 z^m \kappa_1^j$$
$$v_j^m = \rho_2 z^m \kappa_2^j$$

where ρ_1, ρ_2 are constants, $|\kappa_1| \leq 1$, $|\kappa_2| \leq 1$. For these modes to satisfy the boundary conditions, we must have

$$\rho_1 = \rho_2$$
$$\rho_1 z = \rho_1 + \frac{2}{3}\lambda(\rho_1\kappa_1 - \rho_2\kappa_2) + \frac{2}{3}\lambda^2[2\rho_1(\kappa_1 - 1) - \rho_2(1 - \kappa_2)].$$

In matrix notation,

$$\begin{pmatrix} 1 & -1 \\ z - 1 - \frac{2}{3}\lambda\kappa_1 - \frac{4}{3}\lambda^2(\kappa_1 - 1) & \frac{2}{3}\lambda\kappa_2 + \frac{2}{3}\lambda^2(1 - \kappa_2) \end{pmatrix} \begin{pmatrix} \rho_1 \\ \rho_2 \end{pmatrix} = \underline{0}. \qquad (3.3.16)$$

For (3.3.16) to hold, the matrix must be singular, with zero determinant. This gives us the *determinant condition:*

$$z = 1 + \frac{2}{3}\lambda(\kappa_1 - \kappa_2) + \frac{2}{3}\lambda^2(2(\kappa_1 - 1) - (1 - \kappa_2)), \tag{3.3.17}$$

with κ_1, κ_2 and z satisfying the resolvent equations for Lax-Wendroff

$$z = 1 + \lambda(\kappa_1 - \frac{1}{\kappa_1}) + 2\lambda^2(\kappa_1 - 2 + \frac{1}{\kappa_1}) \tag{3.3.18}$$

$$z = 1 + \frac{\lambda}{2}(\frac{1}{\kappa_2} - \kappa_2) + \frac{\lambda^2}{2}(\kappa_2 - 2 + \frac{1}{\kappa_2}). \tag{3.3.19}$$

From the characteristic equations (3.3.18) and (3.3.19), we can solve directly for the κ_j,

$$\kappa_1 = \frac{z - 1 + 4\lambda^2 - \sqrt{(z-1)^2 + 4\lambda^2(2z-1)}}{2\lambda(1 - 2\lambda)}$$

$$\kappa_2 = \frac{z - 1 + \lambda^2 - \sqrt{(z-1)^2 + \lambda^2(2z-1)}}{\lambda(\lambda - 1)}.$$

Substituting these expressions into (3.3.17) and simplifying for a few pages yields the relation

$$\sqrt{(z-1)^2 + 4\lambda^2(2z-1)} + 2\sqrt{(z-1)^2 + \lambda^2(2z-1)} = 0 \quad . \tag{3.3.20}$$

In other words, if there are normal modes satisfying both the interior difference equations and boundary conditions, equation (3.3.20) must be satisfied. Squaring both terms gives

$$(z-1)^2 + 4\lambda^2(2z-1) = 4[(z-1)^2 + \lambda^2(2z-1)]$$

with the solution $z = 1$. However, by substituting this root in (3.3.20), we see that it was introduced by the squaring, and so there are no solutions to (3.3.20). Therefore, our original conservative boundary condition for mesh refinement in space is GKS-stable.

Chapter 4: CLUSTERING and GRID GENERATION

Much of the success of this adaptive mesh refinement algorithm lies in the generation of efficient subgrids. The procedure is to estimate the error at all grid points in level l grids, and flag those points where the error (or some other measure for determining the need for refinement) exceeds a tolerance ϵ. The grid generation procedure creates a new level of grids with mesh width h_{l+1} so that every flagged point is in the interior of a fine grid. The cost of deciding where fine grids are needed, and generating them, is small since it is proportional to the number of coarse grid points. The most expensive cost is the cost per step of integrating the fine grids, which is proportional to their area. Thus we seek to minimize the total area of these refined grids. In addition, we sometimes want to create grids whose coordinate lines are approximately aligned with the solution.

More precisely, when it is time to regrid, a new grid level may be created, an existing level re-created, or no longer necessary existing levels removed. Even if a fine grid should simply be moved over a little, we use the more general approach of creating a new grid, and initializing it with solution values taken from the old refinement before it is deleted.

The outline of the regridding algorithm is as follows. Suppose the current grid structure G has l levels. Based on the error estimates of level l, we might create new grids at level $l+1$. Next, based on estimates from the (larger) level $l-1$ grids, we re-create a new level l, making sure it includes the new level $l+1$. Continuing, the error estimates on level $l-2$ are used to generate level $l-1$, *making sure level l is properly nested*, and so on to the coarsest level. It is important to work from the finest to the coarsest levels, even though this entails the extra work of ensuring proper level nesting. This way, grids are generated using the most accurate error estimates taken from the finest grid at any given point.

Thus, for each existing level of grids, we apply the same procedure to generate the next finer level. This regridding procedure consists of 4 steps:

1) flag points needing refinement
2) cluster the flagged points
3) generate a grid for each cluster

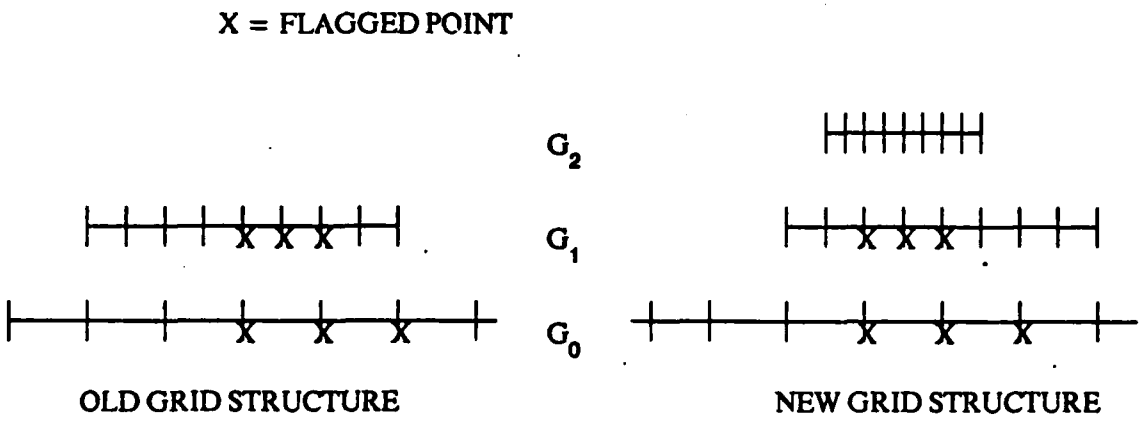


Figure 4.1 1D Regridding Algorithm

4) evaluate, possibly repeat.

Steps (2) and (3) are the difficult steps.

The first step in the algorithm is to identify those grid points at level l which need to be in a finer grid at level $l+1$. In § 2.3 we discussed the use of local truncation error estimates in deciding where these refined meshes are needed. Using the procedure described there, we estimate the error at all grid points at level l, flagging those points $\underset{\sim}{x}$ where $e(\underset{\sim}{x}) > \epsilon$. At this step we also flag grid points in level l grids which are interior to grids at level $l+2$, even if $e(\underset{\sim}{x}) < \epsilon$ based on the level l grid. Since each flagged point at level l will be in a level $l+1$ grid, proper level-nesting is assured.

The second step of the algorithm is the separation of flagged points into distinct clusters. In step three, each cluster will be fit with a fine grid containing all the flagged points of the cluster. In two or more space dimensions, these are difficult steps. In one dimension these steps are simple. We describe the procedure for the one dimensional case here separately. Since in one dimension a grid is just an interval, clustering is trivial and can be done concurrently with grid generation. The leftmost and rightmost flagged points of the coarser grid form the left and right boundary of the new subgrid.

The cluster in this case would consists of all flagged points between and including the leftmost and rightmost flagged points. Possibly, if a long enough gap of unflagged points is found, two or more separate subgrids may be formed instead. The exact definition of *long enough* depends on the size of the buffer zone. After a grid is created, it will be enlarged to include a safety zone around all its flagged points. Recall from § 2.2 that this buffer zone determines how often grids must be examined versus how large they are. The size of this buffer zone is what determines how large the inter-grid spacing should be. Flagged points which are closer together than twice the size of the buffer zone should be in the same grid refinement.

Figure 4.1 illustrates the regridding procedure in one dimension. On the left is the grid structure before regridding, and on the right is the new grid structure. The x's are the grid points which have been flagged with high error estimates. The grid structure is illustrated schematically by drawing each grid separately, instead of superimposing them. We have used a buffer width of one coarse grid point in this illustration.

4.1 Clustering

The clustering algorithm serves two purposes: one is to separate spatially distinct phenomena so that different features of the solution will be in separate grids. The second reason is to subdivide points when a rather large region should be fit with several grids. This situation is illustrated in figure 4.2. If the entire front (represented by the darkened line) were fit with one large grid, it would be an unacceptably large area of refinement. If we had some information about the directional layout of the points, a smart subdivision of the points could be made.

It is very tricky to find a general clustering algorithm that is almost foolproof and is not very expensive. Our approach is to start with a simple algorithm which works in most cases, and try to detect when it does a bad job of clustering. In these cases we use a more expensive algorithm, and try to tailor it to the special cases when the first approach fails. We are lucky to be able to draw on the large literature in pattern recognition and Artificial Intelligence (see, for example, Duda and Hart, [1973]; Hartigan, [1973]). There are algorithms for feature extraction or edge detection as well as more general clustering algorithms with goals similar to ours.

The easiest clustering algorithm, and the first approach that we use, is the nearest neighbor algorithm. The nearest neighbor algorithm forms clusters which are distinguished by having inter-point distances for points in the same clusters smaller than the inter-cluster distances. We start with one point forming a new cluster. Successive points are included in this cluster if the distance from the point to the cluster is less than some specified tolerance, which we usually take to be two mesh widths.

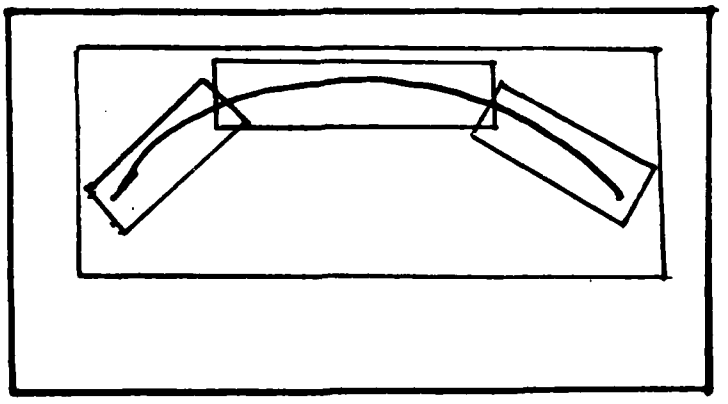


Figure 4.2 Multiple Grids - One Front

The nearest neighbor algorithm is very successful in accomplishing the first goal of clustering, but fails in the second. A single front with high curvature, or interacting fronts which are nearly linear, for example a mach stem configuration, can fool this algorithm, and form one large cluster when several smaller clusters would be better. This occurrence would be signalled by an unacceptable grid evaluation in step 4 of the regridding algorithm. For these cases, a more sophisticated clustering algorithm is needed. We discuss several alternative methods of clustering flagged points, but report that no algorithm has been found which is best in all cases.

Several of the clustering algorithms we use are based on special data structures with certain properties, which we use as a way to organize and process the flagged points. We will include here a short description of the data structures, before describing the clustering algorithms that use them. Shamos and Hoey [1975] and Toussaint [1980] are good references for all the data structures we will mention here.

Several of the approaches to clustering points are based on minimal spanning trees (MST). The MST is the connected acyclic graph connecting all the points so that the sum of the lengths of the edges is a minimum. In this graph, each flagged point is connected to its nearest neighbor. Figure 4.3 contains an example of a minimal spanning tree for

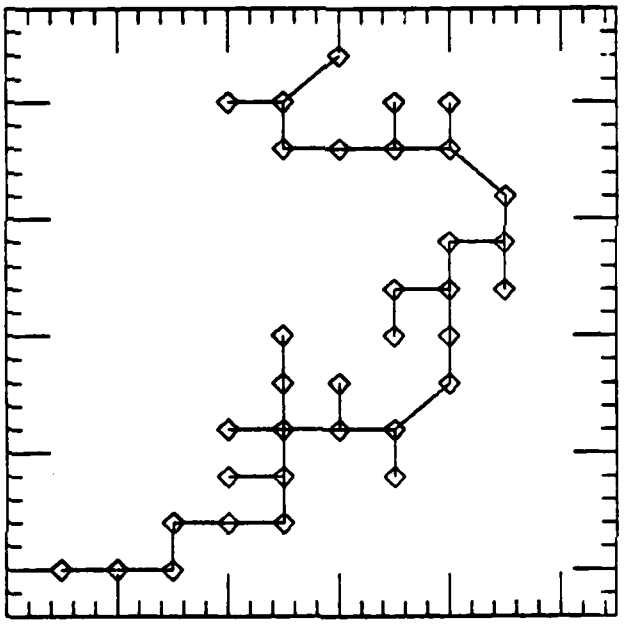
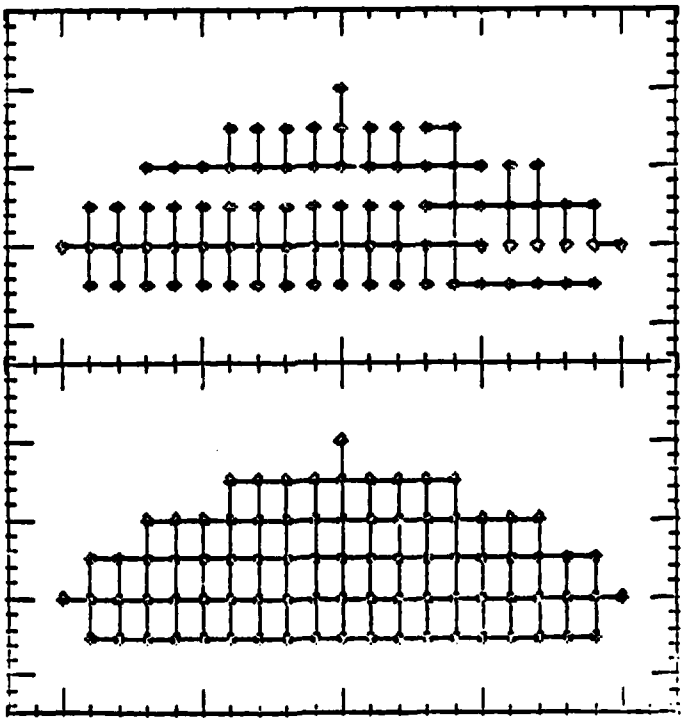


Figure 4.3 Minimal Spanning Tree

that set of points. In general, the MST can be constructed in $O(n^2)$ operations, where n is the number of points in the tree. In two dimensions the MST can be constructed in time $O(n \log n)$, using the Voronoi diagram. Finally, there are algorithms for finding the MST which are more difficult to implement but run in linear expected time, with some assumptions about the distribution of points (Shamos and Bentley, [1981]).

The MST of a set of points is not necessarily unique. If a point has several equidistant nearest neighbors, the MST is constructed by arbitrarily choosing one to make a path in the tree. A drawback of the MST is that this arbitrary quality can lead to trees which don't accurately reflect the layout of the set of flagged points. Figure 4.4(a) shows an example of this. We can summarize the problem in figure 4.4(a) as there being too large a ratio of path length to physical distance between points. One way to fix this problem is to generalize the MST into what is called an All Nearest Neighbors graph. As the name indicates, in this graph a point is connected to *all* its nearest neighbors. In figure 4.4(b) we use the same point set as figure 4.4(a) but connect them with an All Nearest Neighbors graph. As is clear from the graph, we no longer have a tree, since there are cycles in this graph. This data structure also takes $O(n^2)$ operations to compute, but the



(a) MST

(b) All Nearest Neighbors Graph

Figure 4.4

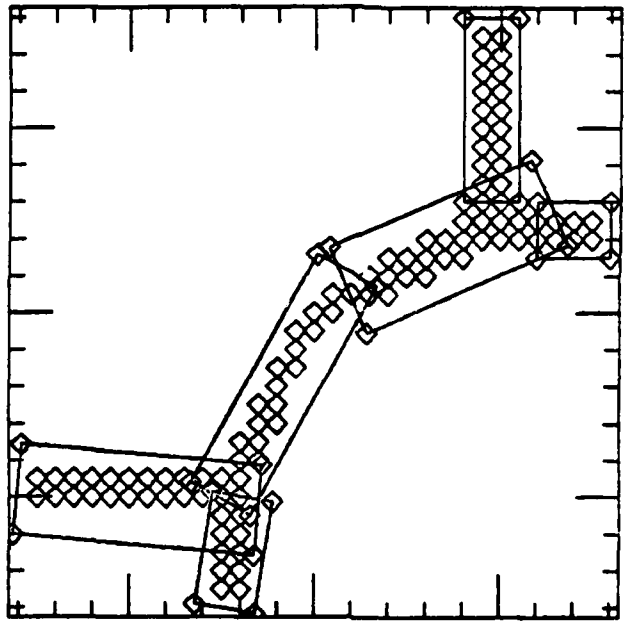


Figure 4.5 Iterative Grid Generation

constant is larger, essentially doubled over that of the MST. We will continue with the rest of the description of the clustering algorithms based on the MST, but caution that in some cases it is preferable to use the nearest neighbor graph instead.

Finally, we remark that in some sense, it is overkill to use such fancy data structures. An approximate all nearest neighbors graph would be sufficient for our purposes. We are experimenting with a straightforward linear time algorithm to generate a graph with enough connectivity to avoid the problems in figure 4.4, and ensure that the next phase of the clustering algorithm still works. We don't have enough experience to report on it at this time.

Using the MST as a way to organize the points, several clustering algorithms are now possible. The first algorithm was initially tested by (Baxter, [1981]). It is an iterative method of grid generation. The method starts with each flagged point in its own individual cluster. The algorithm proceeds iteratively by merging neighboring clusters and immediately fitting a new grid to each cluster until no more clusters can be successfully merged. The grid is fit using the method of §4.2; a successful merge is one which has an acceptable evaluation using the criteria from step 4 of the regridding algorithm, also

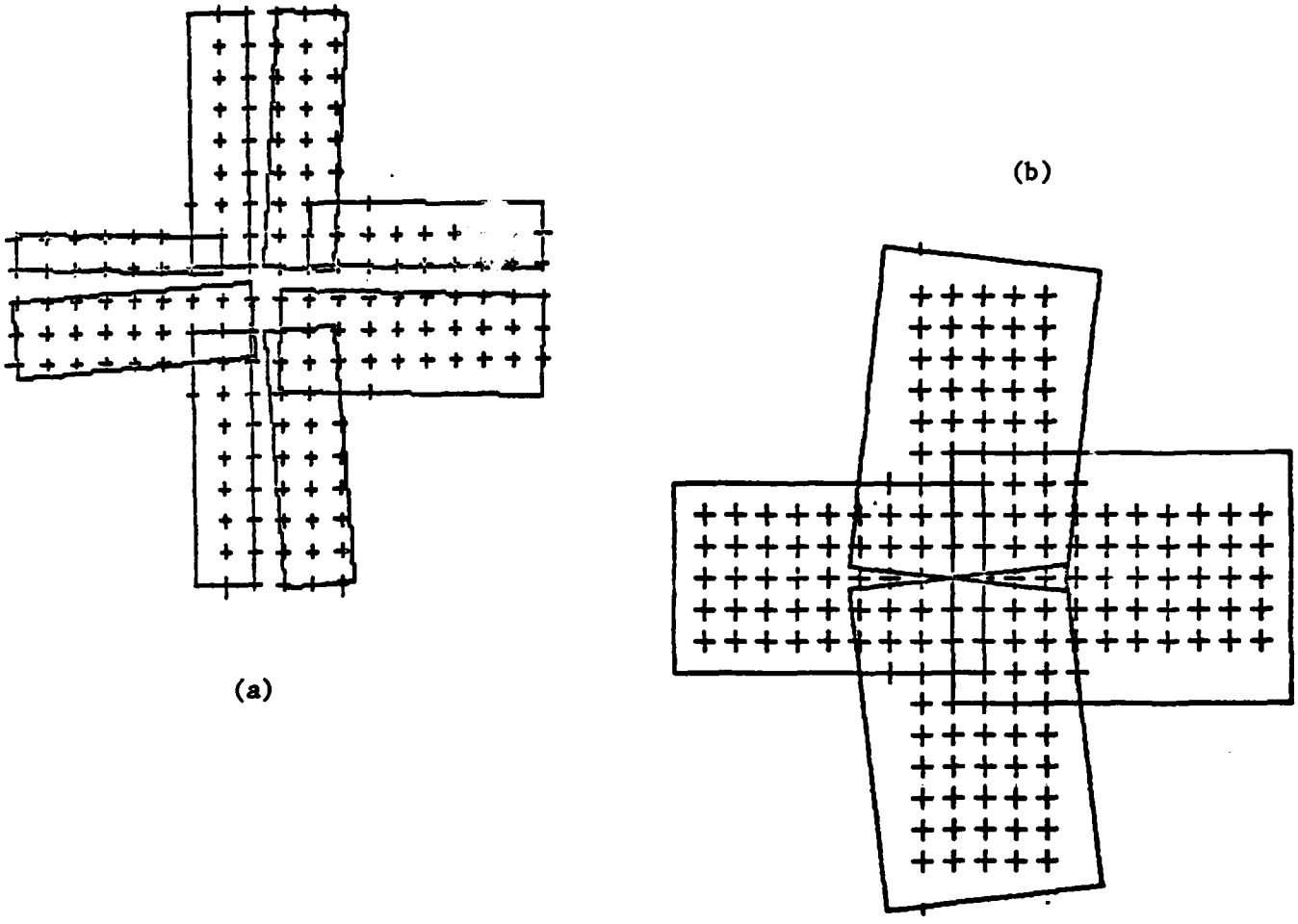


Figure 4.6 Bisection – Merging

described in §1.2. When a cluster contains more than one point, its location is taken to be the mean of the points.

This can be an expensive iteration since calculating the new grid for each intermediate cluster can add up if there are many points in the tree. Also, in this algorithm there are two ways that neighboring clusters can be merged. All neighbors of all points in the cluster can be added at each new iteration, or the neighbors can be added one at a time. The latter possibility, which is naturally the most expensive version, is our most successful algorithm for the combined clustering and grid generation of difficult sets of points. Figure 4.5 illustrates a particular example using this method.

To try to cut down the expense of this algorithm, we have tried a related algorithm which starts with one cluster and bisects, instead of one point per cluster with merges. In this approach, if a grid fit is rejected, we bisect the rectangle that was formed in the long

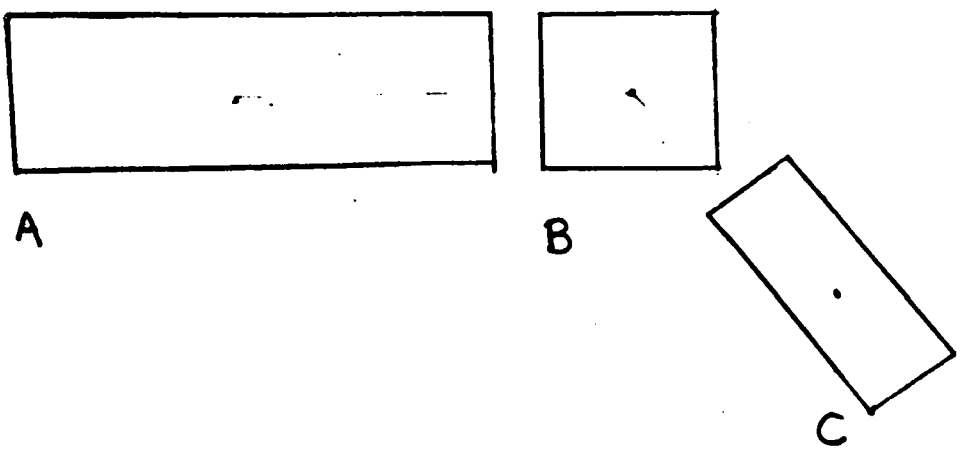


Figure 4.7

direction, sort the points into two clusters based on which half of the rectangle they are in, and try again. The bisection is guaranteed to converge since we bisect the rectangle in the long direction. At that point, when there are typically 10 to 15 small clusters, we start the iterative merging of neighboring clusters just described. Figure 4.6(a) shows the intermediate grids which result from the bisection. Figure 4.6(b) shows the final grids after the merging process.

The advantage of this preprocessing step is that we start merging with only 10 to 15 clusters, instead of perhaps 80 to 100 single point clusters. The disadvantage is that it doesn't work quite as well as the previous algorithm. Simple bisection doesn't use or try to find out any special information about the points, such as their highest density region, direction, *etc*. Also, even an all neighbors graph is not connected enough to represent the relationship between the bisected clusters. In figure 4.7 for example, the means of the rectangles B and C are closer than A and B, but rectangles A and B should be logically connected. This extra connectivity can be added by using yet another data structure, a Relative Neighbor Graph. In this graph, two points are connected if there is no other point which is closer to both of them.

We now present a few alternatives that seem reasonable but have not been tested. These are all algorithms that try to determine some additional properties of the points to use in a smart clustering procedure. For example, one approach based on the MST uses the diameter of the graph to indicate the layout of the points. The diameter of the graph is the length of the longest path between two vertices. Using the MST, the diameter can be found in time $O(n)$ by the following algorithm. Starting with an arbitrary point, find the

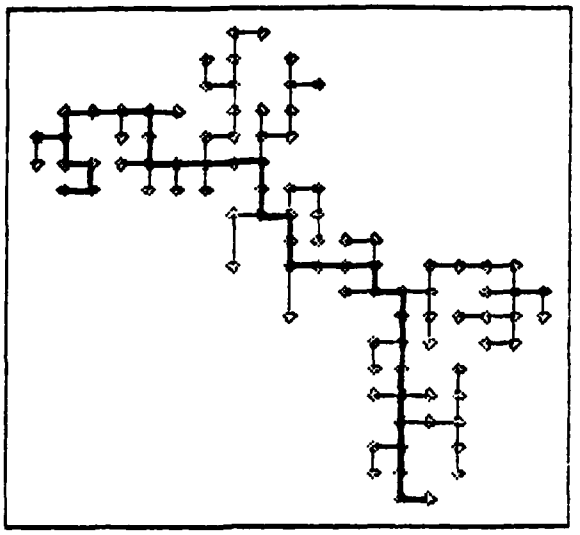


Figure 4.8 Diameter of MST

point a maximum distance away. Now starting from this new point, again find the point a maximum distance away. This last path will be the diameter of the MST. In Figure 4.8 the points are connected in a MST whose diameter is darkened.

In problems with a discontinuity in the solution, if we assume that the points that have been flagged with high error estimates are those along the front, then the diameter of the graph approximates the shape of the front. Clusters can now be formed by segmenting the diameter in an appropriate way. Linear stretches of the diameter can be made into a single grid; regions with high diameter curvature should have smaller segments. Since the diameter itself might be rather jagged, the difficult part of this algorithm is determining the appropriate way to segment the diameter. This seems easier than segmenting the original set of points, however. Several other clustering approaches using the MST are mentioned by (Zahn, [1971]).

There is more information available than we are using right now in this clustering problem. One possibility is to use the actual error estimates themselves at the flagged points. By ridge tracing, that is, connecting points whose error estimate is a local maximum, the general direction of a front can be determined, again assuming that the maximum error occurs along the front. In this way we hope to avoid the cost of creating the MST. As above, the clustering can proceed by grouping linear stretches of the ridge lines into separate clusters.

Both the MST and ridge tracing algorithms have the same difficulty in recognizing direction patterns in a set of points. Algorithms in the pattern recognition and vision processing literature often use iterative techniques (for example for line detection and edge enhancement) to process a scene. Since our algorithm is to be repeated often during a mesh refinement solution, we would hope that a small number of passes through the set of flagged points would suffice for our clustering algorithm. An algorithm with a goal similar to the segmentation into linear stretches described above is used by (Binford and Nevatia, [1975]) to recognize curved objects in the plane. In this algorithm, small groups of points are fit with a least squares cone. If neighboring cones have a small change in the direction of the axis or cross-section, they are considered part of the same curved object. In our case, the change in axis would determine whether or not the combined sets of points can be well described by one aligned grid. The simpler approach of fitting a line to a set of points might do just as well for us however. Statistical measures such as $r = \frac{S_{xy}}{\sqrt{S_{xx} S_{yy}}}$ where the S_{ij} are the second moments about the respective coordinates, can perhaps be used to estimate the linear relationship among the variables. On a different note, early work was done by (Zucker, [1976]) on road identification problems in the field of image processing. This may be a closely related problem.

A final important observation is that once we have good clusters they don't change very fast. If at some initial time an expensive clustering algorithm is used, the same clusters can continue to be used for many time steps. Flagged points can be grouped in the same clusters they were grouped in at the previous step, and only the orientation of a new refined grid need be calculated. If grid points are flagged on a section of a coarse grid not previously refined, the flagged points should be added to the nearest cluster.

Since these clustering algorithms work in close conjunction with the grid generation algorithm, we describe that next.

4.2 Grid Generation

For each cluster of flagged points we want to generate a rectangular grid so that grid lines are in some sense aligned with the points. In addition, the rectangle should have as small area as possible to minimize the work of integrating that grid in time. The algorithm we use for this first determines the orientation of the rectangle, and then the length of the sides needed to enclose the points. We describe our algorithm, and then compare it with some alternatives. For simplicity we present it in two space dimensions, but it generalizes immediately to higher dimensions.

Let A be the n-by-2 matrix of the coordinates of the n flagged points, and A_m the matrix of the same dimension with the x and y coordinates of the mean of the points, (x_m, y_m). The matrix $M^t M = (A - A_m)^t (A - A_m)$, where

$$A = \begin{pmatrix} x_1 & y_1 \\ \vdots & \vdots \\ x_n & y_n \end{pmatrix} \qquad A_m = \begin{pmatrix} x_m & y_m \\ \vdots & \vdots \\ x_m & y_m \end{pmatrix}$$

and

$$M^t M = \begin{pmatrix} \sum_i x_i^2 - x_m^2 & \sum_i x_i y_i - x_m y_m \\ \sum_i x_i y_i - x_m y_m & \sum_i y_i^2 - y_m^2 \end{pmatrix},$$

contains the second moments of the points about their mean. This matrix $M^t M$ determines an ellipse with the same first and second moments as the original set of points (Cramér, [1951]), and so provides a good approximation of the layout of the points. Since $M^t M$ is real and symmetric, it has two real eigenvectors. These eigenvectors are the major and minor axes of the ellipse. We use these eigenvectors to determine the orientation of the rectangular subgrid. This algorithm is invariant under translations and rotations of the points, is extremely simple, and does a great job. It is easy to find the eigenvectors since $M^t M$ is a 2-by-2 matrix. Furthermore, if clustering is done concurrently with grid generation, the first and second moments can be updated instead of recalculating them for every additional point. A similar technique of clustering and fitting ellipsoids has been used by (Gennery, [1979]) for stereo vision processing. The goal of his work is obstacle avoidance for exploring vehicles, for example, the Viking Lander's exploration of Mars.

Once the grid orientation has been determined, the dimensions of the rectangle are calculated to include all points in the cluster. This is the expensive part of the algorithm. We take the dot product of a point with the orientation vectors for every point in the

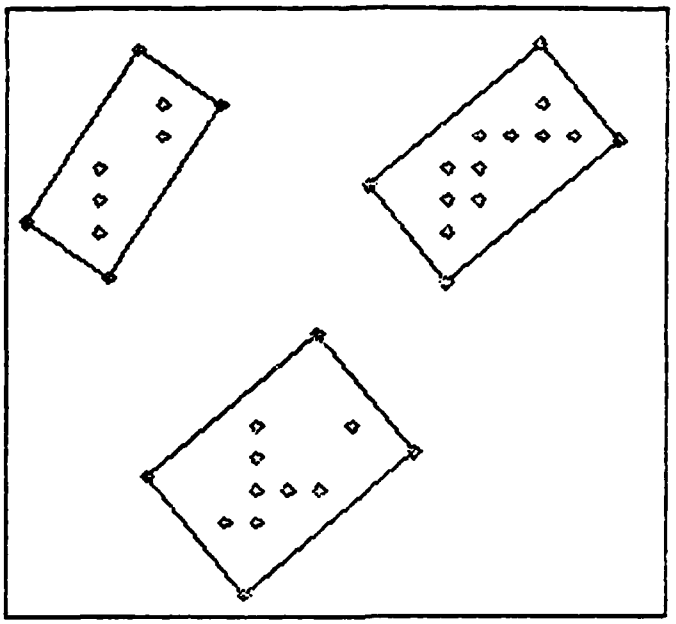


Figure 4.9 Rectangle Orientation

cluster. (Some of this work can be avoided if the convex hull of the set of flagged point is kept. For iterative algorithms, there are also ways to update the convex hull of a set of points for the addition of new points.) Once the dimensions of the rectangle are calculated, a buffer zone of predetermined size is added around the rectangle perimeter to complete the new subgrid. Figure 4.9 shows the grids this algorithm generates for three different clusters. We have used a buffer zone of one coarse mesh width in this illustration.

It turns out that this algorithm is related to a total least squares fit of the data points in the following way (Golub and Van Loan, [1980]). Suppose we look for a linear fit to the data with a line through the mean of the points, $(y - y_m) = m(x - x_m)$, such that

$$\left\{ \begin{bmatrix} x_1 - x_m \\ x_2 - x_m \\ \vdots \\ x_n - x_m \end{bmatrix} + \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} \right\} m = \begin{bmatrix} y_1 - y_m \\ y_2 - y_m \\ \vdots \\ y_n - y_m \end{bmatrix} + \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \end{bmatrix} \tag{4.2.1}$$

We determine the slope of the line m so that the Frobenius norm of the perturbation vectors $\underset{\sim}{e} = (e_1, e_2, \ldots, e_n)^t$ and $\underset{\sim}{r} = (r_1, r_2, \ldots, r_n)^t$ is a minimum. We rewrite (4.2.1) as

$$\left([\underset{\sim}{x} - \underset{\sim}{x}_m \,|\, \underset{\sim}{y} - \underset{\sim}{y}_m] + [\underset{\sim}{e} \,|\, \underset{\sim}{r}] \right) \begin{bmatrix} m \\ -1 \end{bmatrix} = \underset{\sim}{0}$$

For the smallest perturbation, take the perturbation matrix $C = [\,\underset{\sim}{e} \mid \underset{\sim}{r}\,]$ to be the smallest singular value of the matrix $[\underset{\sim}{x} - \underset{\sim}{x}_m \mid \underset{\sim}{y} - \underset{\sim}{y}_m]$. This matrix is just the matrix $A - A_m$ of the flagged points. The solution vector $(m, -1)^t$ is the singular vector corresponding to the smaller singular value. This singular vector of $A - A_m$ is one of the eigenvectors of the matrix M^tM above, and so both derivations give the same rectangle orientation. By using the total least squares derivation, we see that we are finding the slope of the line through the mean of the points which minimizes the sum of the squares of the distances from each point to the line.

One of our goals in grid generation is to create subgrids with as small area as possible. This algorithm does remarkably well in that. We explain this heuristically in the following way. Suppose the sides of the rectangle are oriented in the directions of the orthonormal unit vectors $\underset{\sim}{r}_1, \underset{\sim}{r}_2$. The length of the side parallel to $\underset{\sim}{r}_1$ is

$$\max A\underset{\sim}{r}_1 - \min A\underset{\sim}{r}_1$$

where again we have

$$A = \begin{pmatrix} x_1 & y_1 \\ \vdots & \vdots \\ x_n & y_n \end{pmatrix},$$

and similarly for $\underset{\sim}{r}_2$. The *max* and *min* are taken componentwise over the vector. We know that the subgrid coordinate line will go through the mean of the points. If the points are clustered so that there are no extreme outlying points, (which we certainly hope the clustering algorithm does), we get that the length of a side is

$$\max A\underset{\sim}{r}_1 - \min A\underset{\sim}{r}_1 \approx 2\,\|A\underset{\sim}{r}_i\|_\infty.$$

Thus we consider the maximum norm problem of minimizing the area of the enclosing rectangle,

$$\min_{\substack{r_1, r_2 \\ r_i r_j = \delta_{ij}}} \|Ar_1\|_\infty \|Ar_2\|_\infty. \tag{4.2.2}$$

If we approximate (4.2.2) using the 2-norm this problem can be easily solved. We use the fact that

$$\|Ar_1\|_2^2 \|Ar_2\|_2^2 = r_1^t A^t A r_1 r_2^t A^t A r_2$$

but for orthonormal vectors r_1, r_2,

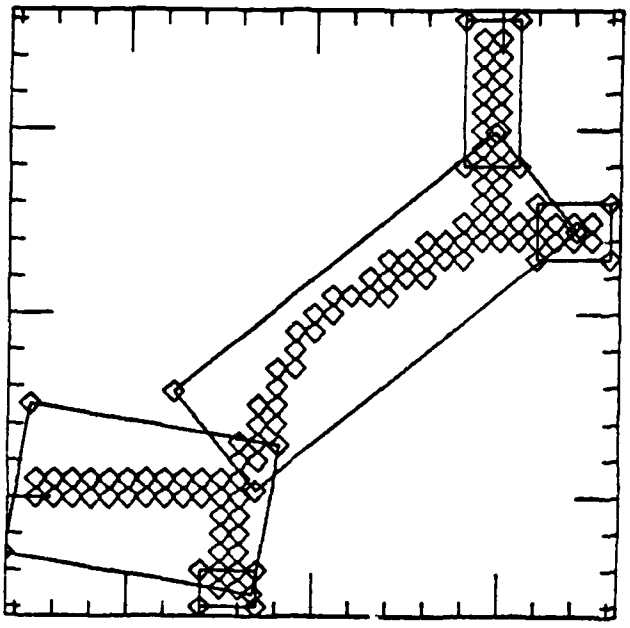
$$r_1^t A^t A r_1 + r_2^t A^t A r_2 = constant.$$

The problem again becomes that of minimizing $\|A\underset{\sim}{r}\|_2$ over unit vectors $\underset{\sim}{r}$, giving the same orientation vectors that the moment generated ellipse gives. So the area of the enclosing rectangle is related to the area of the minimum enclosing rectangle as the 2-norm is related to the maximum norm. In general there can be a large difference between those two norms. However, if we use a smart clustering algorithm, this won't be the case.

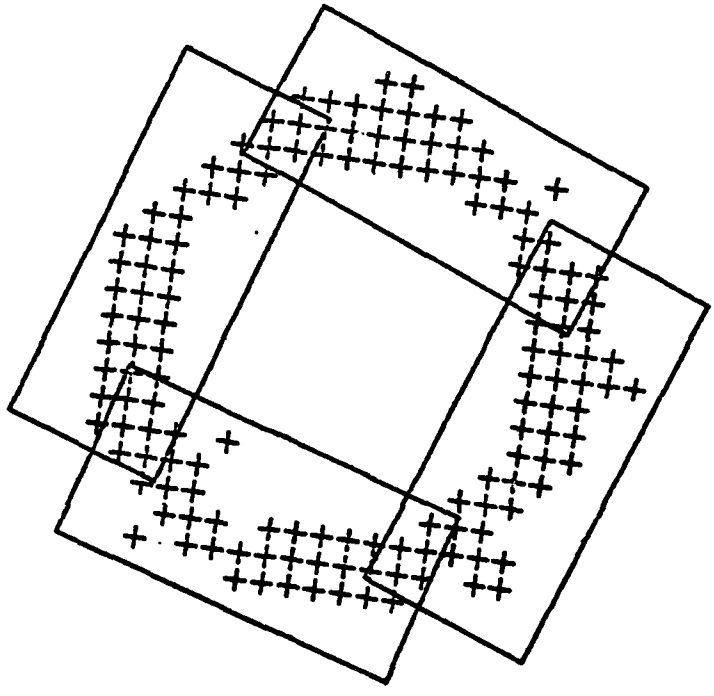
In two dimensions, it is computationally feasible to actually construct the minimum area rectangle enclosing a set of points. We will briefly describe how to do this, and compare it with our procedure above. To construct a spanning rectangle, first find the convex hull of the set of points. In two dimensions, this can be done in $O(n \log n)$ operations, where n is the number of flagged points to be enclosed (Graham, [1972]). The next step makes use of a theorem by (Freeman and Shapira, [1975]) proving that the minimum area rectangle has a side collinear with the convex hull. It remains to check each side of the hull for the rectangle with minimum area. For each line segment on the hull, the area of the spanning rectangle can be determined in time $O(h)$, where h is the number of vertices on the convex hull, by carefully figuring which vertices on the hull anchor the rectangle. Finally, choose the rectangle with the minimum area.

This algorithm doesn't generalize easily to higher dimensions, where the convex hull takes $O(n^2)$ operations to compute. Even in two dimensions, it is much more expensive than using the ellipse method. Since the minimum spanning rectangle depends only on the convex hull of the set of points, it will be aligned with the points only if the clustering algorithm aligns the points first. In experiments comparing the minimum area spanning rectangle with moment generated rectangles, the latter typically has 5 to 10% larger area than the former.

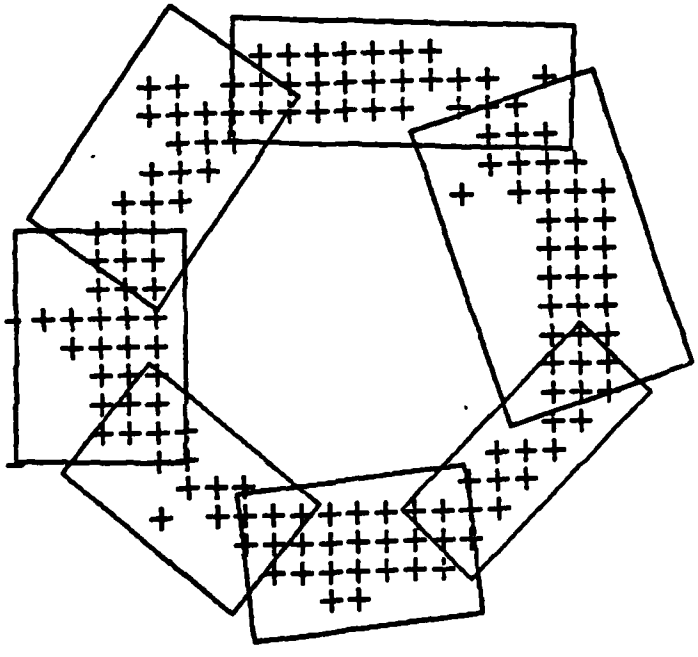
Returning to the regridding procedure, the final step in the algorithm evaluates the goodness of fit of the new grids. Our practical criterion for measuring this uses the fraction of the area of the rectangle which is unnecessarily refined. This can be estimated quickly by taking the ratio of flagged points to the total number of coarse grid points in the new fine grid. If this ratio falls below a cutoff tolerance, typically between $1/2$ and $3/4$, we recluster the points and try again. The pictures in figure 4.10 illustrate the different subgrids that are formed using the *volume cutoff* parameter indicated. In Figure 4.10(a) the grids were generated with a cutoff tolerance of .45. In figure 4.5 the same



(a)



(b)



(c)

Figure 4.10 Efficiency of Subgrids

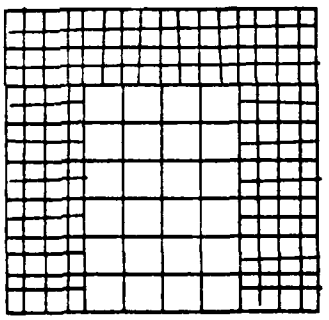
set of points and same iterative grid generation procedure were used with a tolerance of .85. In figure 4.10(b) and (c) the bisection procedure was used with tolerances of .5 and .75.

Two additional points will complete the discussion of the regridding algorithm. As outlined above, this algorithm creates only one new level of refinement for each invocation. For time dependent boundary conditions, if no assumptions are made on the smoothness of the boundary data, an incoming wave might need several new levels of refinement for it to be well resolved. To handle this case, the error estimation and grid generation procedure can be re-applied to a newly created fine grid at the boundary to see if even finer new grids are needed.

Finally, we mention that the initial grid creation at time $t = 0$ employs a slightly different strategy than the regridding procedure used at later times. Only at this time can we take advantage of the initial conditions specified with the problem. For example, when a level 1 refinement is created, it is initialized using the initial conditions rather than interpolation. The error estimation and regridding procedure can then be applied again on the level 1 grids to see if even finer subgrids are needed, and so on, until a pointwise error estimate $e(\underset{\sim}{x}) < \epsilon$ holds at all points.

Chapter 5: DATA STRUCTURES

The data structures that we use in adaptive mesh refinement are what makes this entire approach feasible. Simply the question of how to manage the grid and solution data that is adaptively generated is difficult, without even considering the potential costs of storage and run-time overhead. Previous adaptive strategies are greatly limited because of their failure to confront the lack of data management facilities in Fortran. This is evident, for example, in some of the multi-grid implementations (Brandt, [1976]) for solving elliptic pde with recursively nested fine grids. In common implementations, refinements are allowed ony between two coiumns on the underlying coarse grid. This kind of fixed grid representation prohibits horseshoe-type fine grids of the following type, with for example two refinements in a given row.



Finite element programmers are very aware of the data management problem, and are usually more advanced in their data structures. Several successful adaptive finite element programs are found in the literature, for example Bank [1981] and Babuška and Rheinboldt [1978]. Rheinboldt and Mesztenyi [1980] are experimenting with a complicated tree data structure to represent nonuniform two dimensional rectangular meshes in their adaptive calculations. Simpson [1981] has a general purpose two dimensional mesh verification procedure with linked lists to keep track of the vertices, element incidences, etc.

In our adaptive mesh refinement strategy, data structures are needed to keep track of each grid, the grid's relation to other grids in the hierarchy, and the approximate solution

on each grid. There are several places during every time step where this information is needed. Boundary values for a fine grid come from either a neighboring or parent grid. In turn, the fine grids update the coarser, parent grids. Every several time steps the entire grid structure may change. Grids are formed or removed, and the new grid relations must be determined dynamically. In designing the data structure for our adaptive mesh refinement program, we have tried to minimize the cost of the operations that will most frequently be done. We will first describe the data structure we use in one dimension. A generalization of this data structure is what we use in two or more dimensions.

Recall from § 2.1 the nesting requirement for one dimensional mesh refinement: each fine grid must be entirely contained in a coarser grid at the next level. This leads naturally to a tree data structure. We represent each grid by a node in the tree. Subgrids are considered the *offspring* of their coarser, parent grid. Siblings are subgrids within the same coarser grid. If fine grids are at the same level of refinement with different parents, we call them neighbors. Technically, we use an ordered tree data structure where each node can have multiple descendents. In this representation, we see that a node will have multiple descendents if the coarse grid corresponding to that node has several fine grids contained in it. In this one dimensional case, it is also possible to order the nodes using the coordinate value of the left-most grid point in the associated grid.

The following definition of a tree is due to (Aho, Hopcroft and Ullman, [1976]). A tree data structure is a directed acyclic graph with the following properties:

1) There is one "root" vertex, where no edges enter.
2) Each vertex except the root has exactly one entering edge.
3) There is a unique path from the root to each vertex.

Figure 5.1 shows a grid structure and its related tree structure.

All the grid-to-grid operations described above have an information flow which follows path links in the tree. The only non-standard link in the tree is for the neighbor pointer we described above. This thread is indicated by the dashed line in figure 5.1. This additional link makes it easy to implement the operation of taking one integration step on all grids which are at the same level of refinement.

Because this tree structure can grow or shrink dynamically, some form of dynamic storage allocation is needed, both for the grid information in each node, and the solution storage for each grid. Since Fortran does not provide such a facility, the storage management must be explicitly provided by the mesh refinement program. We keep a

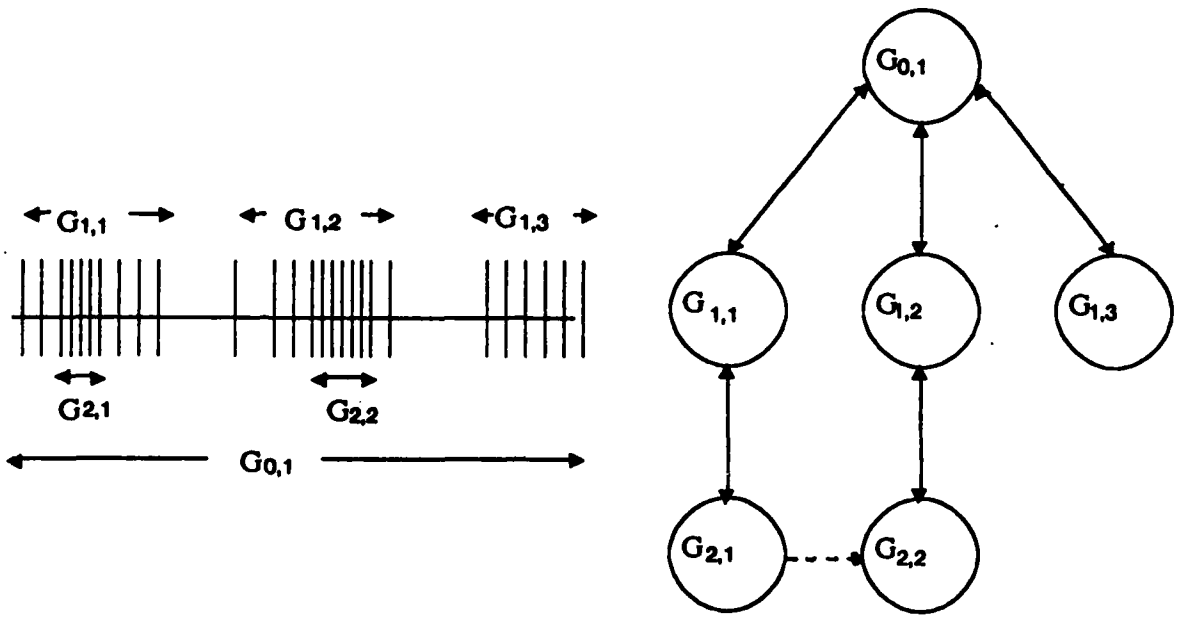


Figure 5.1 $1D$ Data Structure

linked list of free nodes which are assigned to newly created grids and reclaimed when a grid is removed. Sometimes when the regridding procedure is called, it is to move grids on only the finest levels. The coarser level grids stay fixed. In this case, the top half of the tree will remain as is. A new bottom half will be generated, and the two halves connected.

Although the number of nodes in the tree vary, we would like each node to contain a fixed amount of information to represent each grid. Since the number of offspring of each node is variable, the tree is implemented with each node having an offspring pointer only for the first descendent, with one sibling pointer per node to connect the rest of the subgrid nodes, (Knuth, [1968]). Each node in the tree, however, is connected to its parent grid by its own parent link. The tree in figure 5.1 is implemented as indicated in figure 5.2.

All the nodes can be accessed using standard tree traversal algorithms. We simply mention two here. A depth – first traversal would list each grid followed by its leftmost

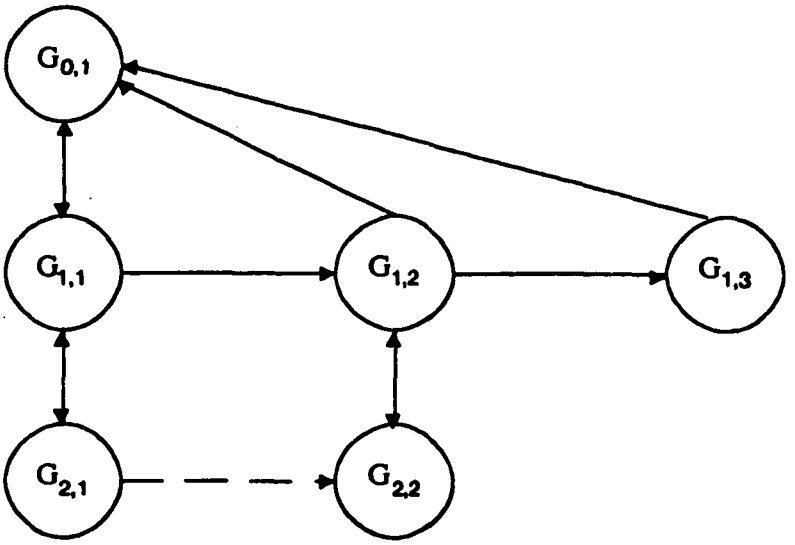


Figure 5.2 1D Implementation

subgrids. A breadth – first search would list all grids at a given level of refinement, and then all grids at the next level of refinement. These traversal algorithms are used, for example, to output the grid structure, or find a grid containing a given point. (Knuth, [1968]) is an excellent reference for this.

A grid then is characterized by the following pieces of information stored in each node of the tree.

1) Grid location.
2) Number of grid points.
3) Level in tree.
4) Offspring pointer.
5) Sibling pointer.
6) Parent pointer.
7) Pointer to the next grid at the same level.
8) Time to which this grid has been integrated (real and integer).
9) Index into main storage array where approximate solution values are stored.

If we set the unlikely maximum value of 100 possible fine grids, each with 10 pieces of information, the 1000 words of storage overhead for the grid information is still unimportant compared with the storage needed for the solution itself.

The two dimensional version of this data structure is more intricate that the one dimensional version since in two dimensions a grid can be partially nested in more than

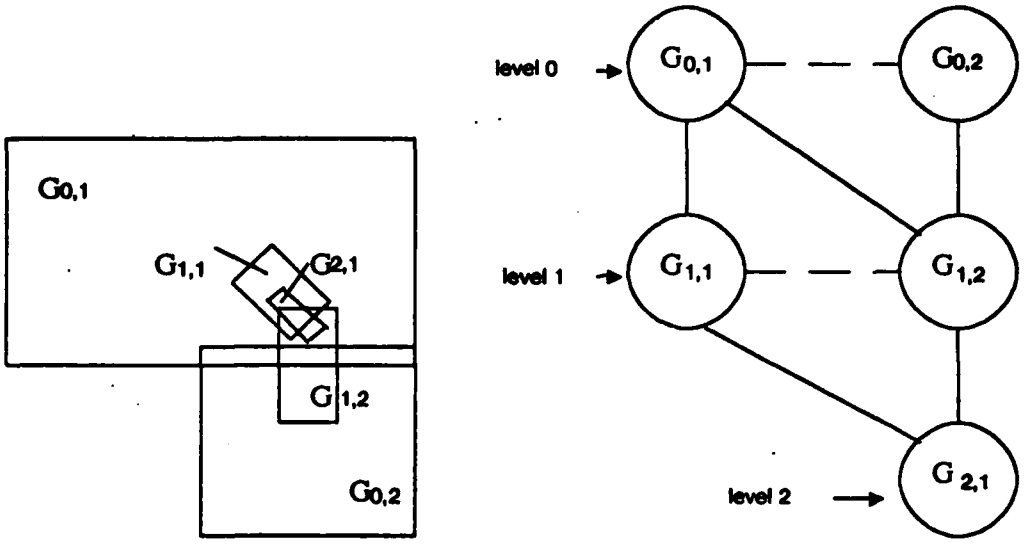


Figure 5.3 $2D$ Data Structure

one coarser grid. An additional complication is that grids at the same level of refinement can intersect. Finally, in two dimensions there is the possibility of having several base grids $G_{0,j}$ in an initial domain specification. We have generalized the one dimensional data structure to account for these differences in the following way. We start with an n-tuply rooted tree, where each of the n root nodes correspond to a component grid in the coarsest mesh. Technically, a tree with more than one root node is called a forest, which is simply a collection of trees. Next, since a subgrid can have many possible parent grids, we replace this single slot of information in each node with a pointer to a short linked list of parent grids. Lastly, we add to the information for each grid a pointer to another linked list of intersecting grids at the same level of refinement. Schematically this data structure is illustrated in figure 5.3.

We also consider, but have not yet implemented, adding two more pieces of information to the data structure. We could speed up both the updating procedure, where a fine grid updates the solution of its parent grid, and the internal boundary value

procedure, where a fine grid gets boundary values from either a neighboring fine grid or one of its parent grids, by storing some extra information in the grid structure. Since the overhead associated with these two procedures is a lower order amount of work than the integration of the interior values, this is not a substantial fraction of the run-time. The extra information could appreciably speed the run-time overhead for these operations however.

The two dimensional data structure therefore has several places where a linked list is used instead of a single piece of information. We implement this by keeping another linked list of free space available for use by any of the grids. The space is reclaimed when the grid is deleted. If every grid used, for example, 20 slots from this linked list to specify several parent grids, overlapping fine grids, boundary values etc, and there were 20 grids in existence at the same time, 400 words of storage would still be negligible. So all in all the grid storage and the extra linked list storage represent a very small amount of overhead.

The final data structure used in our mesh refinement program manages the large array which is the storage area for the solution values on all grids. For problems in p space dimensions, we use a p-dimensional array. This storage area is managed as a linked list of used and available blocks of storage. For vector rather than scalar problems, we use a $p+1$ dimensional array where the extra dimension is the number of variables in the problem. When a grid is created, contiguous storage space is reserved from the list of free blocks using a first-fit algorithm (Knuth, [1968]). In this algorithm, the list of free blocks of storage is scanned until a large enough block is found. The requested space is allocated, and the unused portion returned to the list. Reclaimed space, which occurs when a grid is no longer necessary, is inserted back into the linked list of free space. The list of free blocks is sorted on the index into the storage array. For quick memory access, space is never allocated in a circular fashion across the array boundary: all memory allocation must be contiguous. A compaction, or garbage collection, routine could be included to provide for cases where there is enough total but non-contiguous space available in this array to satisfy a storage request. However, as Knuth [1968] reports, if storage is too fragmented to service a request, then compaction usually adds only a few more transactions before space is exhausted. A routine which would allow the user to restart with a larger memory area would be more useful.

We conclude with a philosophical remark. One of the best features of this algorithm is its use of advanced data structures techniques. Such data structures are often used in other branches of computer science, but rarely in numerical analysis. This is no doubt due to the structure of Fortran, which does not encourage such use. We note that although all the data structure management routines had to be specially coded in the mesh refinement program, they make up only several hundred lines of Fortran. They are not difficult to implement, although the necessity of doing so initially detracts from the ease of trying new methods such as this one. Just as graph-theoretic techniques have become standard fare for solving spare matrix problems, other areas of numerical analysis might similarly benefit from the wise use of data structures for problems previously thought to be too difficult.

Chapter 6: STEADY STATE PROBLEMS

Our objective in looking at steady state calculations is to be able to perform transonic flow calculations in two or even three space dimensions. These computations are typically done using highly irregular coordinate systems, where it is difficult to control the mesh spacing over the entire domain. Our mesh refinement approach could use component coarse grids, for example, a body-fitted coordinate system around the airfoil, and a Cartesian grid for the far field to avoid this problem. Furthermore, our method has all the machinery needed to resolve the standing shocks in these problems by using grid refinements.

To develop some experience in, and tailor this algorithm for steady state computations, we have been working on a scalar model problem in one space dimension. We will briefly describe the changes needed for the algorithm to solve steady state equations. Since a common solution method for these problems is marching in time, this fits into the framework of our algorithm very easily. A few components change. For example, the error estimation procedure is a little different. We also have modified the algorithm to use implicit finite difference methods. This is especially important since for reasons of efficiency, artificially large time steps are used in marching to steady state, and so the method must be implicit to maintain stability.

The 1D scalar transonic equation we use for our model problem is taken from (Embid, Goodman, Majda, [1982]). The equation is

$$\begin{aligned} u_t + \frac{1}{2}(u^2)_x &= a(x)u \\ u(x,0) &= u_0(x), \end{aligned} \tag{6.1}$$

on $0 \leq x \leq 1$. For the hyperbolic problem (6.1) we can specify two boundary conditions $u(x=0) > 0$ and $u(x=1) < 0$. In regions where the solution u is smooth, at steady state we have $u_t = 0$ or

$$uu_x = a(x)u,$$

so by specifying a we specify u.

On the other hand, if u is discontinuous, the Rankine-Hugoniot condition for the shock speed s is

$$s[u] = \left[\frac{u^2}{2}\right]$$

where the bracket notation denotes the jump in the quantity. This gives

$$s = \frac{u_L + u_R}{2}, \tag{6.2}$$

where u_L and u_R are the solution values to the left and right of the discontinuity respectively. The entropy condition requires $u_L > u_R$, which says that the characteristics flow into the shock, not out of it.

Once the function a is chosen, the boundary conditions $u(0)$ and $u(1)$ determine the solution. Let

$$u_L(x) = \int_0^{x_J} a(x)\,dx + u(0)$$

$$u_R(x) = -\int_{x_J}^1 a(x)\,dx + u(1).$$

The location of the discontinuity at x_J is determined at steady state from (6.2). When the shock speed s is 0, we must have

$$u_L(x_J) = -u_R(x_J).$$

For our model problem, we have chosen the function $a(x)$ so that the steady state solution is

$$u(x) = \begin{cases} e^{-25(x-1/4)^2}, & x < x_J \\ e^{-25(x-1/4)^2} - 2u_L, & x > x_J \end{cases}$$

where we choose the shock location at $x = .56$. The solution is illustrated in figure 6.1. The function $a(x)$ that determines this solution is

$$a(x) = -50(x - 1/4)e^{-25(x-1/4)^2}.$$

We have chosen this peaked function to be able to illustrate the different refinement levels. The Gaussian is steep enough to need at least one level of refinement to resolve the peak sufficiently. At the discontinuity, the maximum allowable number of refinement levels will be used.

We have modified the multiple grid integration algorithm in the following ways. In the original mesh refinement algorithm, the error estimation and grid generation happens every few time steps, depending on the wave speed. In this case however, we iterate until steady state has been reached on a given grid configuration. Then we estimate the

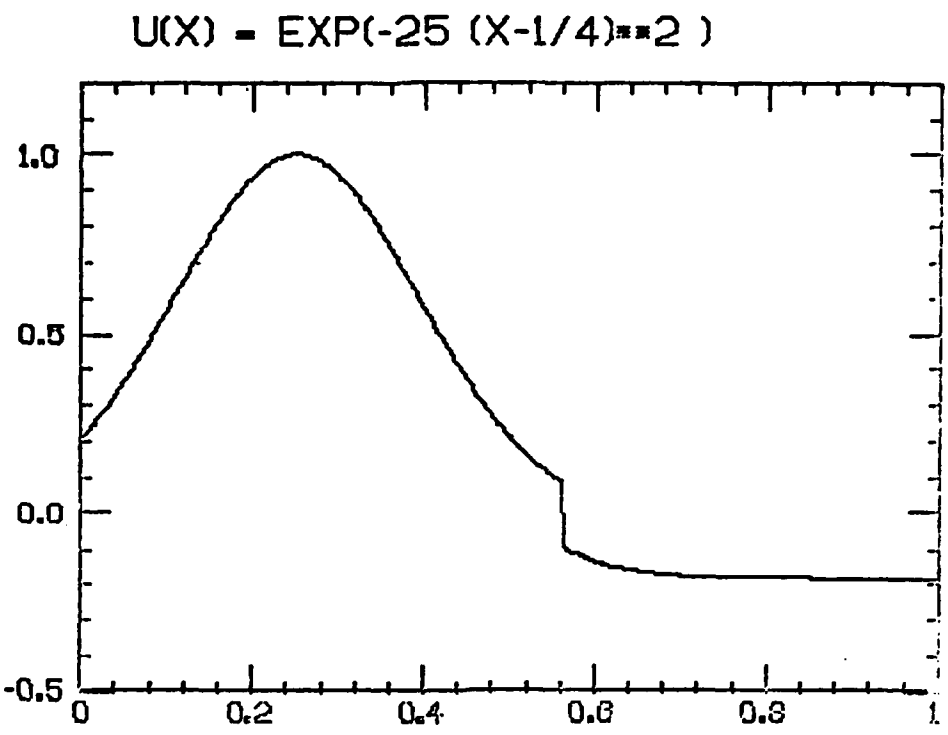


Figure 6.1

error and add another level of refinement if necessary. Since in these calculations the (arbitrary) initial conditions might have very little to do with the final solution, it seems pointless to estimate the error and put fine grids around a transient solution feature.

When steady state is reached, if a new finer grid level is created, it will be initialized by interpolation from the coarser grid. In this sense, our algorithm resembles the nested grid approach found in multigrid methods, for one example, of *bootstrapping* the solution. It is cheaper to first compute the solution on the coarse grid and therefore have a good starting guess for the fine grid solution, than to solve directly on the all grids simultaneously. Multigrid methods also bootstrap in the other direction, by having the coarse grids help in the solution *process* itself. We have not experimented with this, although it seems a natural thing to consider given the nested grid structure of the algorithm. We point out a further difference between multigrid methods and our adaptive method. Multigrid methods adaptively decide on the solution process, by deciding on which grid to iterate next. The grids themselves are usually fixed, although this is not necessary in principle. In our adaptive method, the grids themselves are adaptively generated. If the estimated error on the finest grid is still too large, we can make even finer grids, and they do not necessarily cover the entire domain. Of course if they do, it

makes little sense to have the overhead of the underlying grids unless they are used for something, which multigrid methods do.

How to arrange the integration algorithm on several grids is a good question deserving consideration. We have done the easiest thing, which is to directly apply the integration algorithm in §2.2 to steady state computations by substituting the word iteration for integration. For example, in a computation with a coarse grid and a fine grid refined by a factor of 2, we iterate once on the coarse grid for every two iterations on the fine grid. This is inefficient, however, since we make no adjustments now for iterating more on one grid if the other is almost converged.

It is important to realize that one grid cannot converge independently of the other, since there are finite difference equations for each grid which use values from the other. Since the iteration is performed separately on each grid, this interaction is evident in two ways. For the fine grid, boundary values for the next iteration are taken from the coarse grid. The coarse grid values are updated from the fine grid. This cycle must also be at steady state for the solutions on each grid to be at steady state.

We examine this more closely in considering the marching methods we use in solving this problem. We derive our explicit marching algorithms by using forward Euler differencing in time, and either first or second order upwind differences in space. Since the solution has a discontinuity, we would like the difference equations to be in conservation form. At the sonic point, where u changes sign and it is unclear which direction is upwind, we use the Engquist-Osher switch (Engquist and Osher, [1981]). (The methods are discussed more fully in (Embid, Goodman, and Majda, [1982]).) This gives us the method

$$v_j^{n+1} = v_j^n - \frac{\lambda}{2} D(v_j^{n2}) + ka(x)v_j^n, \tag{6.3}$$

For the first order method the difference D is defined by

$$D(v^2) = \Delta^+(\psi(v)v^2) + \Delta^-(\phi(v)v^2) \tag{6.4}$$

where

$$\psi(v) = \begin{cases} 1, & v > 0 \\ 0, & v < 0 \end{cases}$$

and

$$\phi(v) = 1 - \psi(v).$$

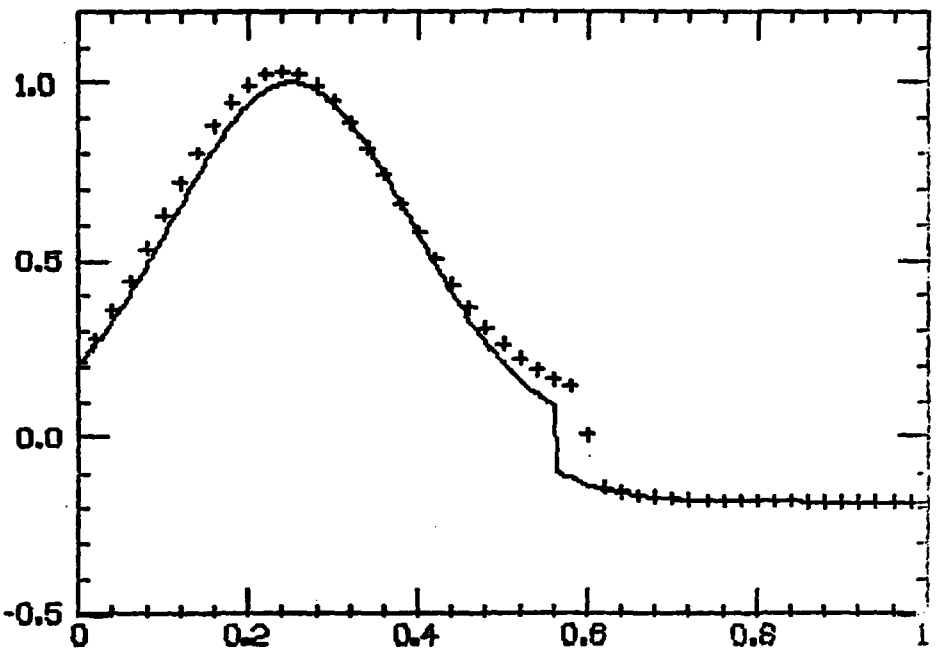


Figure 6.2 1^{st} Order Differences

In (6.4) we use the notation Δ^+ and Δ^- to mean the forward and backward difference operators, without division by h. For the second order explicit method we use instead

$$D(v^2) = \Delta^{++}(\psi(v)v^2) + \Delta^{--}(\phi(v)v^2) \tag{6.5}$$

where Δ^{++}/h is the second order one sided approximation to the first derivative,

$$\Delta^{++}(v_j) = -\frac{1}{2}v_{j+2} + 2v_{j+1} - \frac{3}{2}v_j,$$

and similarly for Δ^{--}.

For the first order method, the stability region limits the time step so that $k/h = \lambda < 1$, and for the second order method this limit is $\lambda < 1/2$. The explicit methods are thus extremely slow to converge on each grid. Additionally, it is slow because of the cycle between fine and coarse grids mentioned above. The implicit verison of our marching algorithm is derived by using backward Euler differences in time and the same difference operators in space. We get

$$v_j^{n+1} = v_j^n - \frac{\lambda}{2}D(v_j^{n+1^2}) + ka(x)v_j^{n+1}. \tag{6.6}$$

This is a nonlinear system where we take one Newton step for every iteration. (The efficiency of this method is also a question deserving consideration.) To use the implicit method with refined grids, it is necessary to use interpolation boundary conditions to get boundary values for the fine grids. At convergence, this is identical to using the coarse mesh approximation method.

We have computed the solution using the first and second order explicit methods, and the first order implicit method. We first compare the number of iterations needed to converge when there is only one coarse grid. For a typical example, we use a mesh spacing of $h = .2$ for a total of 51 grid points. We stop iterating when the maximum update $|v^{n+1} - v^n| < 10^{-5}$.

method	λ	*#iterations*
1^{st}order explicit	.8	1296
2^{nd}order explicit	.4	1783
1^{st}order implicit	.8	1313
1^{st}order implicit	50	46
1^{st}order implicit	500	29

When the second order method converges, it is much more accurate than the first order method. This is especially evident by looking at the shock location. For this grid spacing, $h = .2$, the first order methods get the shock location at around $x = .6$. The second order method gets it almost exactly right. Figures 6.2 and 6.3 illustrate this. The symbols are the computed solution, and the solid line the exact solution.

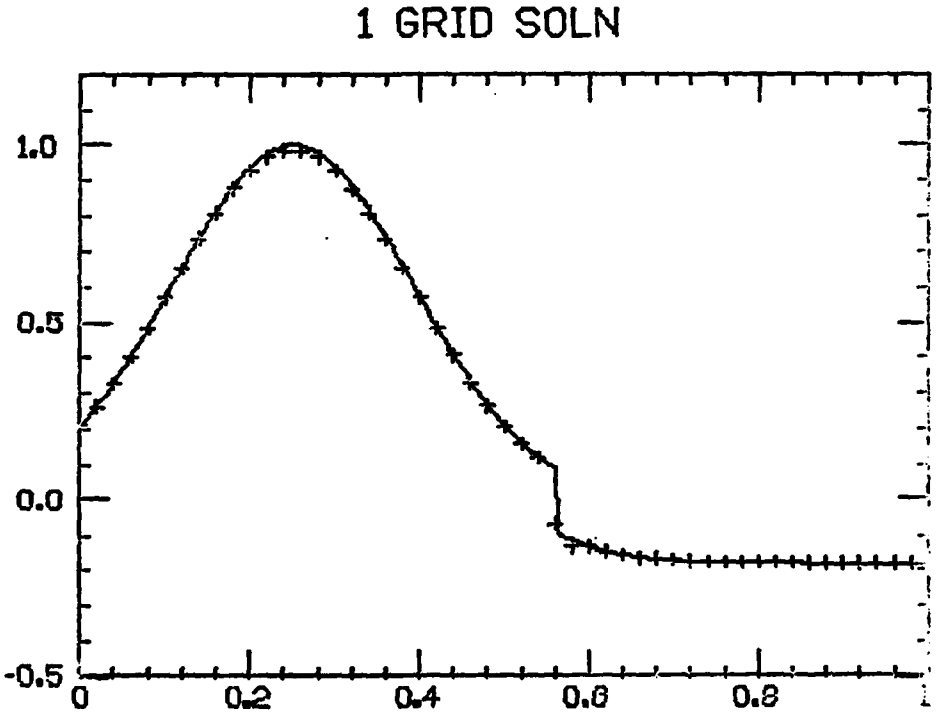


Figure 6.3 2^{nd} Order Differences - 2^{nd} Order B.C.

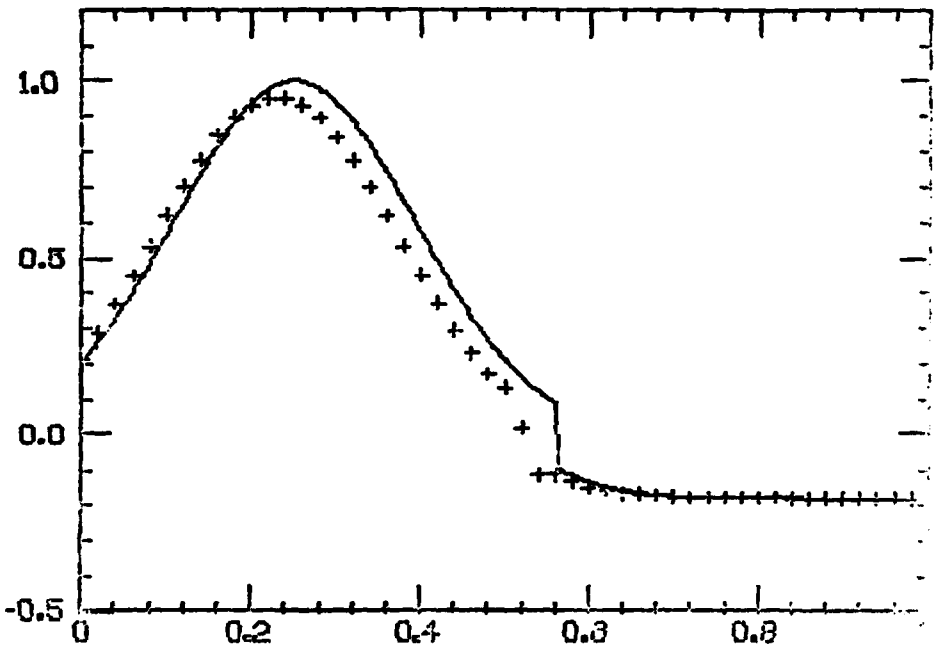


Figure 6.4 2^{nd} Order Differences - 1^{st} Order B.C.

We point out that the second order method is especially sensitive to the choice of the initial solution vector, and for some choices would not converge. This is due to our choice of boundary conditions. Since at either boundary the stencil for this method needs 2 points to one side to determine the third, an additional boundary condition formula is needed. We used central differences for the point next to the boundary, which is also second order accurate. These boundary conditions appear to be only marginally stable. We have also tried using the first order upwind scheme for the point next to the boundary. Figure 6.4 shows what a big difference this makes, even though this lower order accurate boundary condition is only applied at two points.

Before looking at the solutions with refined grids, we finish describing the algorithmic changes to our mesh refinement strategy. It remains to discuss the error estimation procedure. We would still like to use Richardson estimates of the local truncation error, since this will give a procedure that is independent of the equation and the method of differencing. If we write the difference equations as

$$v^{n+1} = (I - kQ)v^n$$

then at steady state

$$v^{\infty} = (I - kQ)v^{\infty},$$

or $Qv^{\infty} = 0$. The error we are interested in estimating is due only to the spatial discretization. If u is the exact solution to the pde, the local truncation error τ is defined

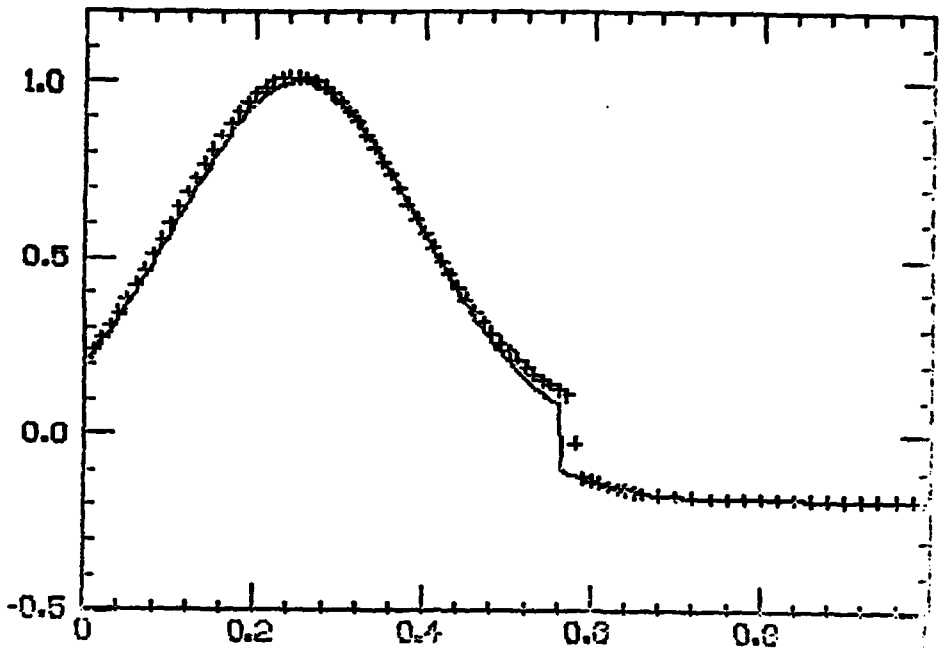


Figure 6.5

by $\tau = Qu$. Suppose

$$Qu = \tau = ch^q + O(h^{q+1}),$$

where c depends on spatial derivatives of the solution u. If we form

$$\begin{aligned} Q_{2h}u &= (2h)^q + O(h^{q+1}) \\ &= 2^q\tau + O(h^{q+1}) \end{aligned}$$

we have our error estimate. We would like to do this in as automatic a way as possible. Suppose we compute

$$w = (I + kQ_{2h})v^\infty,$$

and form the pointwise difference

$$\frac{v_j^\infty - w}{k} = Q_{2h}v_j^\infty = 2^q\tau + O(h^{q+1}) \tag{6.7}$$

We thus get the error estimate at the cost of one integration step, followed by computing the discrete difference in time. This is basically the first step of a deferred correction procedure.

This same method works for implicit methods as well. At steady state we have

$$(I + kQ)v^\infty = v^\infty,$$

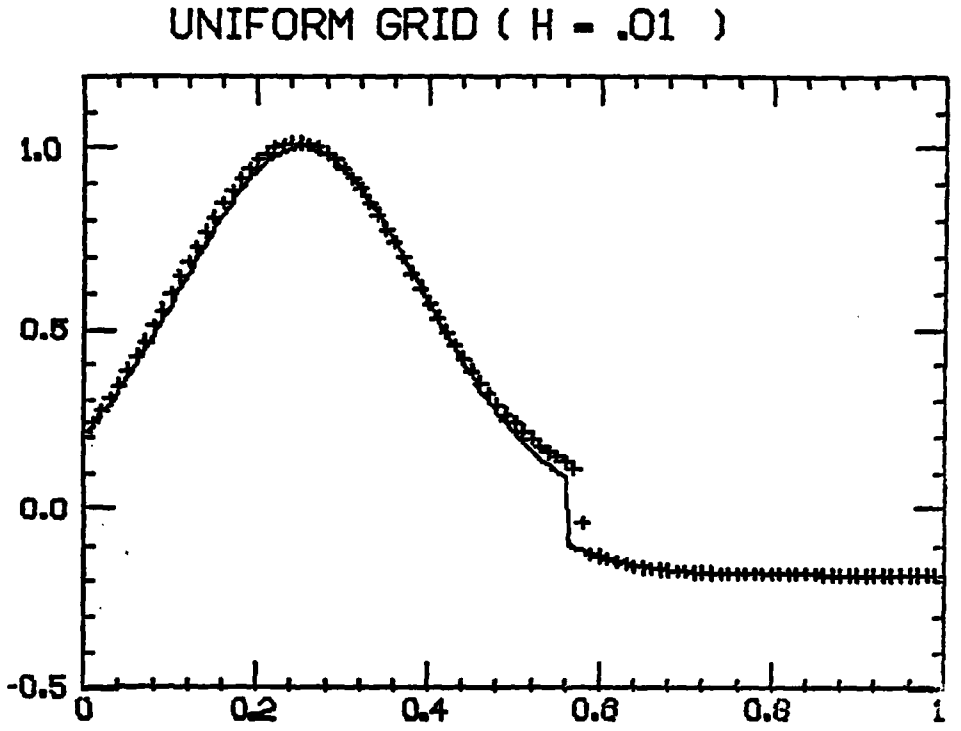


Figure 6.6

and we compute

$$(I + kQ_{2h})w = v^{\infty}.$$

If k is small,

$$(I + kQ_{2h})^{-1} = (I - kQ_{2h} + O(k^2))$$

so by forming the difference

$$\frac{w_j - v_j^{\infty}}{k} = Q_{2h}v_j^{\infty}$$

we again get an estimate of the leading term of the local truncation error.

We show the results of some computations with the first order method using mesh refinement. Figure 6.2 already shows the results of a uniform grid calculation with mesh spacing $h = .02$. In figure 6.5 we show the results using 2 grid levels, $h = .02$ and $h = .01$. We use an error tolerance of .075 and refine by a factor of 2. Compare the results with figure 6.6 where we have a uniform fine grid run using the finer mesh width $h = .01$. We have approximately the same accuracy in both runs. Nothing is gained by going to 3 levels. For further improvement, we would have to reduce the error tolerance as well.

Chapter 7: COMPUTATIONAL EXAMPLES

In this chapter we will present some numerical experiments in one and two dimensions to illustrate how our adaptive mesh refinement algorithm works. Our procedure for comparing results is the following. We solve each problem with mesh refinement with a specified coarse grid, a maximum number of levels of refinement, and an error tolerance which is the criterion we use in deciding to refine a grid. We compare the solution to that obtained on a uniform grid with first the coarsest, then the finest (if possible) mesh spacing used in the mesh refinement calculation. We measure the computer time without I/O costs (when possible) and the error in the solution. In all these cases, we compute the 2 norm of the error at only the coarse grid points. In one dimension, this means we compute

$$\|\text{error}\|_{2_c} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \text{error}(x = ih_c)^2},$$

and similarly in two dimensions. For the fine grid this means we compute the error only at every h_c/h_f grid points. The one dimensional examples were run on an IBM 370/168 using the Fortran H Extended compiler, optimization level 3. The two dimensional examples were run using the same compiler on an IBM 370/3081.

Example 1 Shock Tube Problem

In this example, we compute the solution to the shock tube problem in one space dimension. This Riemann problem is taken from Sod [1978] where it was used to compare a number of methods for solving gas dynamics problems. The initial conditions are chosen so that the solution contains a shock, contact discontinuity, and a rarefaction wave. The problem is:

$$u_t = f(u)_x$$

for $0 \le x \le 1$, and $0 \le t \le .15$, where

$$u = \begin{pmatrix} \rho \\ \rho u \\ e \end{pmatrix} \qquad f = \begin{pmatrix} \rho u \\ \frac{m^2}{\rho} + p \\ \frac{m}{\rho}(e+p) \end{pmatrix}.$$

Here, ρ is the density, e is energy per unit volume, u is velocity, and m is the momentum density ρu. The equation of state is $p = (\gamma - 1)\rho\epsilon$, where we take $\gamma = 1.4$ and ϵ is the internal energy per unit mass $\epsilon = e - \frac{1}{2}\rho u^2$. The initial conditions we use are

$$u(x,0) = 0$$

$$p(x,0) = \begin{cases} 1.0, & \text{if } x < 0 \\ 0.1, & \text{if } x > 0 \end{cases}$$

$$\rho(x,0) = \begin{cases} 1.0, & \text{if } x < 0 \\ 0.125, & \text{if } x > 0 \end{cases}$$

The integration method we use is the two-step Lax-Wendroff scheme. The coarse grid step size is the same as in Sod [1978]. Reflecting boundary conditions are used. Hopscotch artificial viscosity is used to smooth the solution. The coefficient of the hopscotch artificial viscosity was the same for all but the uniform coarse grid run. For the coarse grid this led to too much smearing and so a smaller coefficient is used.

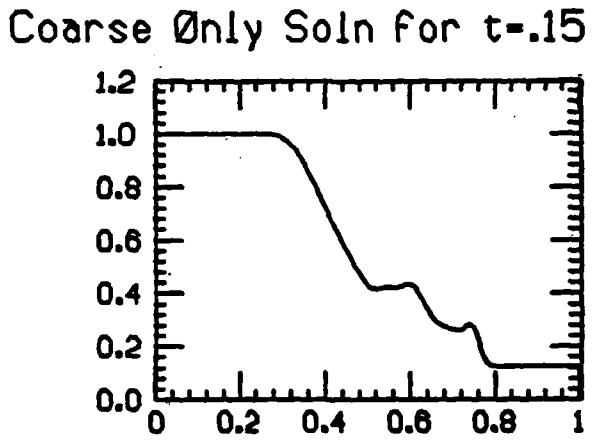
The parameters for the mesh refinement calculation are shown below.

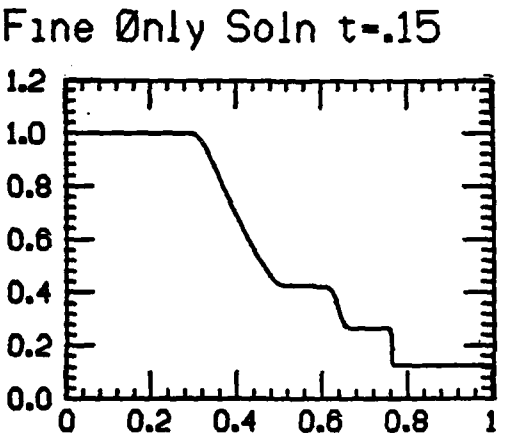
buffer size	4
grids move every	5 steps
error tolerance	.0005
refinement ratio	10
λ	.1

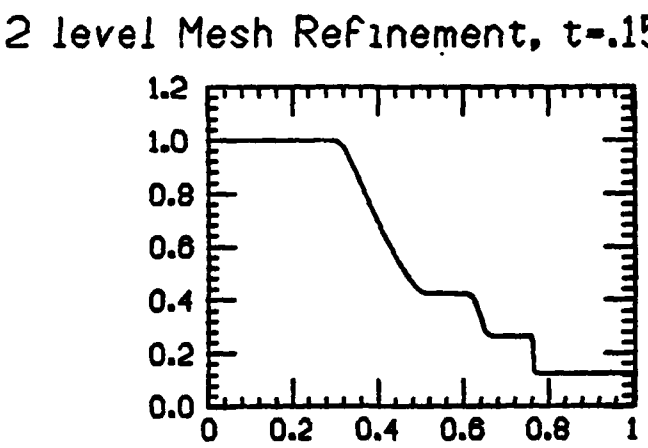
In figures 7.1 and 7.2 and table 7.1 we show the results of our tests. Figure 7.1 shows the solution, and figure 7.2 the plots of the errors. In this experiment, we have done two uniform grid runs with a mesh spacing of .01, and .001. We compare this to a two level mesh refinement run with a refinement ratio of 10, and a three level refinement run, which means the mesh spacing on the finest grid is .0001. During the calculation, the amount of the coarse grid which was refined varied from 20% to 70%.

The most interesting results here are that in less than one fourth the time, mesh refinement is able to calculate a solution which is as accurate as the uniform fine grid calculation. As we see in figure 7.2, most of this error is due to the smearing of the corners of the rarefaction and contact discontinuity. To improve the calculation of the contact discontinuity, a better method than Lax-Wendroff should be used.

Table 7.1 shows the computation time and the $\|.\|_{2_c}$ errors for the four computations. There are a large number of floating point underflows, each of which is handled very







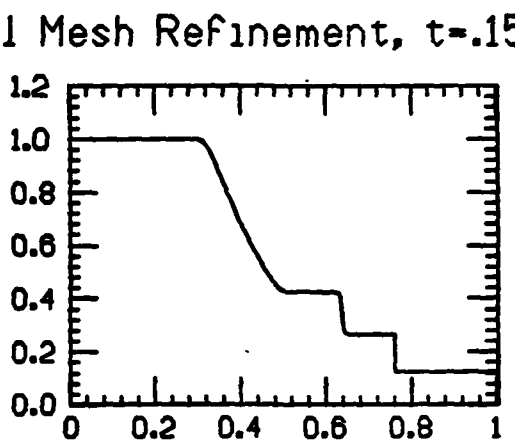


Figure 7.1 Solutions for Riemann Problem

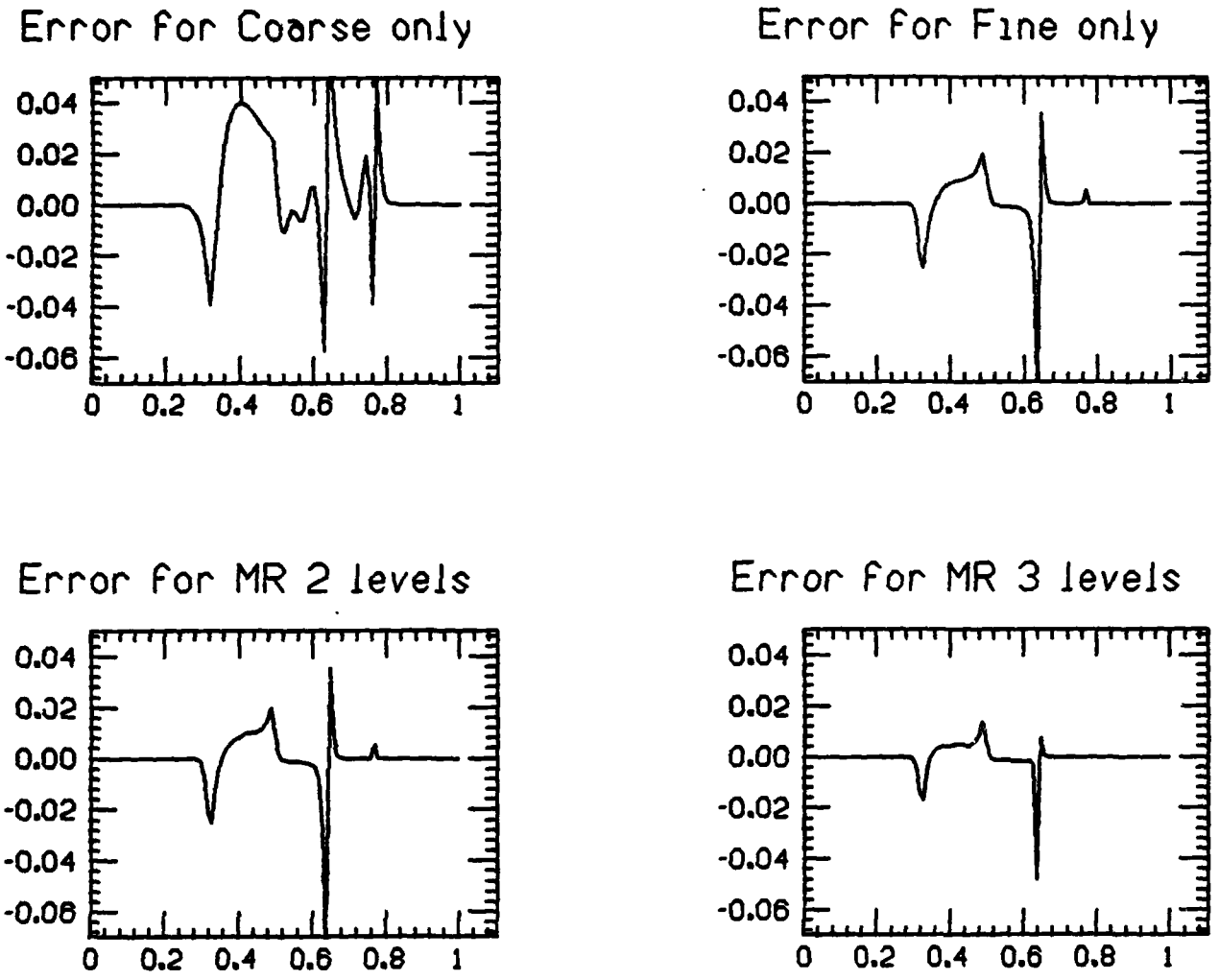


Figure 7.2 Errors for Riemann Problem

inefficiently as an interrupt to the processor. This skews the timings and so the correct asymptotic ratio of fine to coarse timings of 100 is not observed.

An important feature to note here is that while a refinement by h_c^2 (which is a factor of 100) is possible by using mesh refinement (given our computing resources), a computation with a uniform h_c^2 grid would not be possible (it would take about 347×10^2 seconds, or about 9 and a half hours).

method	h	$\|\text{error}\|_{2_c}$	time (secs)
coarse	0.01	$2.14E-2$	7.36
fine	0.001	$1.15E-2$	347
MR 2 lev	h_f =0.001	$1.15E-2$	79.9
MR 3 lev	h_f =0.0001	$6.53E-3$	591

Table 7.1 Results of computations for 1-D nonlinear example.

Example 2 1-D Linear Problem

We use this example to illustrate some properties about the error estimator we use and to demonstrate how we equidistribute the errors in the computation. This example is in some ways the most disadvantageous for mesh refinement because the cost of integration is so small, making the overhead in managing the grids proportionately more important.

The problem is

$$u_t = -u_x$$
$$u(x,0) = .5\sin(4\pi x) + \exp(-200(x-.5)^2)$$
$$u(0,t) = .5\sin(4\pi(x-t))$$

We compute on the interval $0 \leq x \leq 2$, and $0 \leq t \leq 1$. The major feature of the solution to this problem is a right moving pulse on top of a right moving sine wave. For a coarse grid of mesh size of approximately 1/40, the pulse is not accurately represented on the coarse grid while the sine wave is. Our criteria for refinement applies refinement only to the pulse. We can then compute the solution to the same degree of accuracy over the pulse as well as over the sine wave (see figures 7.3 and 7.4).

Again, the method we use is Lax-Wendroff. The inflow boundary is specified as above; the outflow boundary is handled with characteristic extrapolation. The parameters for the mesh refinement calculation are given below:

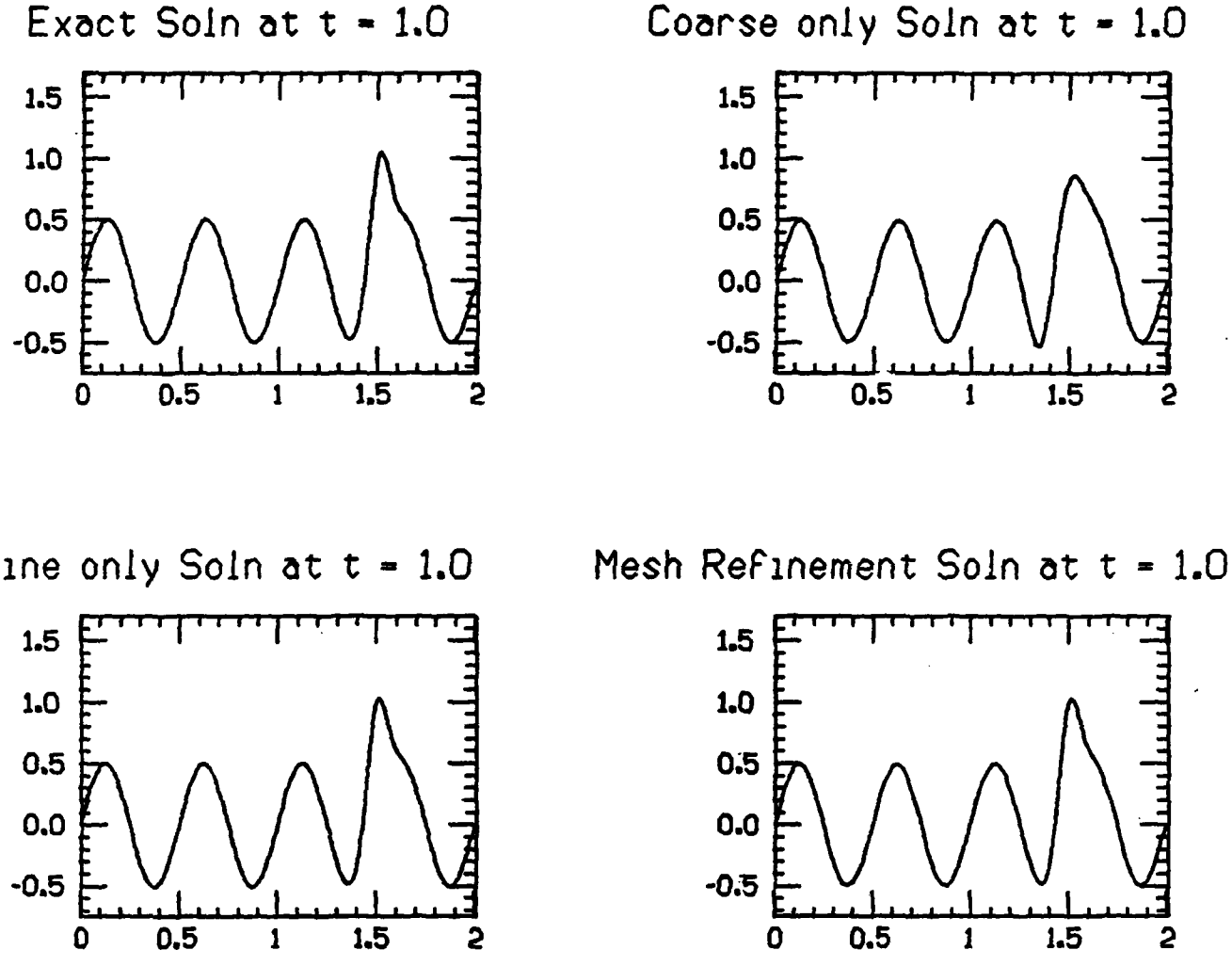
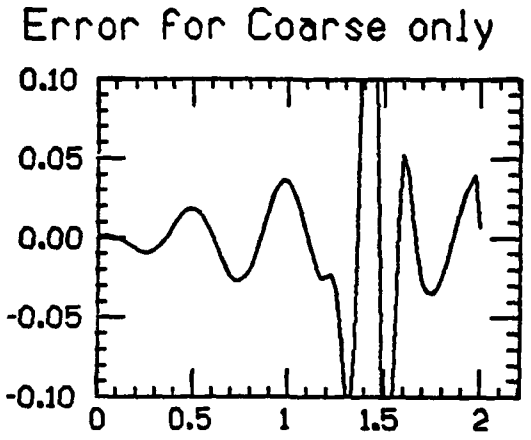
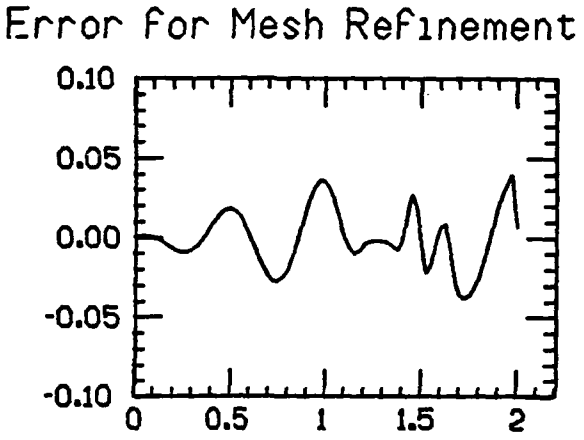


Figure 7.3 Solutions for 1D Linear Problem





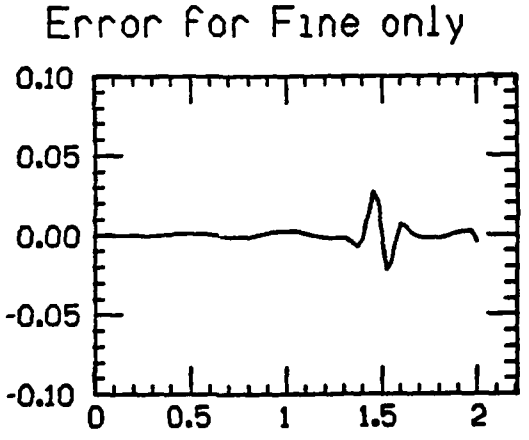


Figure 7.4 Error for 1D Linear Problem

buffer size	5
grids move every	5 steps
error tolerance	.005
refinement ratio	4
λ	.8

Figures 7.3 and 7.4 and table 7.2 show the results of our tests. First, the plots of the pointwise error illustrate how the sine wave is well resolved on all grids, including the coarsest. However, the pulse is resolved only on the fine grids. (notice the loss of height, as well as the "shoulder", in the coarse grid computation).

The second point to note is that the graph of the error for mesh refinement has nearly equidistributed the error. Also, the errors on the coarse grid of the mesh refinement calculation match the uniform coarse grid computation, and the errors on the uniform fine grid match the fine grid of the mesh refinement calculation.

Table 7.2 shows the times and the $\|\cdot\|_{2_c}$ error for the four computations. The behavior of the error reflects the equidistribution of the error by mesh refinement; mesh refinement reduces the error only around the pulse. The time for the coarse grid computation is still dominated by execution overhead; further computations show the correct asymptotic behavior of the computation time as a function of grid size.

method	h	$\|\text{error}\|_{2_c}$	time (secs)
coarse	$1/40$	$5.83E-2$	0.033
fine	$1/160$	$5.48E-3$	0.320
MR	$h_f = 1/160$	$1.77E-2$	0.183

Table 7.2 Results of computations for 1-D linear problem

Example 3 2-D Rotating Cone

The rotating cone problem has been used by Gottlieb and Orszag [1977] to compare numerical methods for convection problems. The problem is:

$$u_t = yu_x - xu_y$$

$$u(x,y,0) = \begin{cases} 0, & \text{if } (x-\frac{1}{2})^2 + 1.5y^2 > \frac{1}{2} \\ 1 - 2((x-\frac{1}{2})^2 + 1.5y^2), & \text{if } (x-\frac{1}{2})^2 + 1.5y^2 < \frac{1}{2} \end{cases}$$

on a rectangular domain $-1 \leq x \leq 1, -1 \leq y \leq 1$, and $0 \leq t \leq 3.375$. The solution to this problem is a cone with elliptical base, which rotates counterclockwise about the origin. We integrate the solution until the cone is approximately halfway through the first revolution.

We use Lax-Wendroff as the integrator. The boundary conditions are zero inflow and first order extrapolation outflow. The parameters for the mesh refinement calculation are:

buffer size	2
grids move every	8 steps
error tolerance	.001
refinement ratio	4
λ	.25

This problem serves for two dimensions the same purpose as the linear problem serves in one dimension. In particular, we illustrate the algorithm described in §4 for subgrid generation. In figure 7.6 we show snapshot views of the location of the one subgrid in this problem at various intermediate time steps.

The results of this computation are what we would expect. The mesh refinement computation achieves about the same error as the uniform fine grid in about one sixth of the time. In this example, we can also get a rough estimate of the overhead of the method. A uniform fine grid run should asymptotically take 64 times the computer time for the coarse grid run. This is because the grid is refined by 4 in both coordinate directions, and there is a factor of 4 for refinement in time as well. In this rotating cone problem, roughly 12% of the grid is refined during the computation. Adding 12% of the cost of the fine grid run to the coarse grid time gives an estimated time of 78 seconds, and so 9 seconds of the total mesh refinement computation time are spent on other things besides integrating the grids. This is roughly 12% of the computing time, most of which is spent estimating the error and generating the subgrids. However, the entire run costs only 16% of the cost of a run on the uniform fine grid with the same accuracy.

Notice that in the mesh refinement computations, the wake behind the cone is greatly reduced over the wake in the coarse grid computations. This shows that the moving grid is correctly computing the solution and keeping the coarse grid computation under

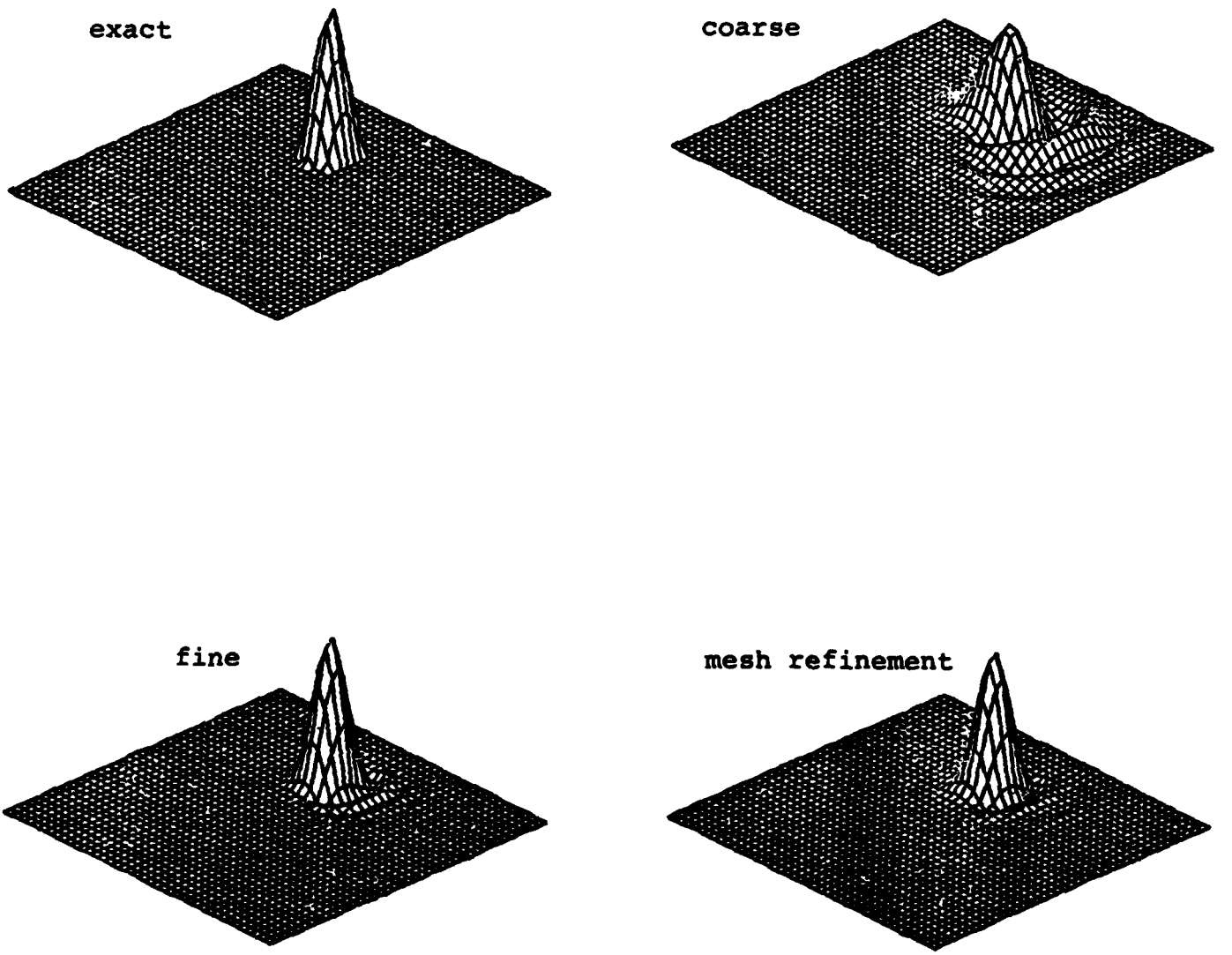


Figure 7.5 Solutions for Rotating Cone Problem

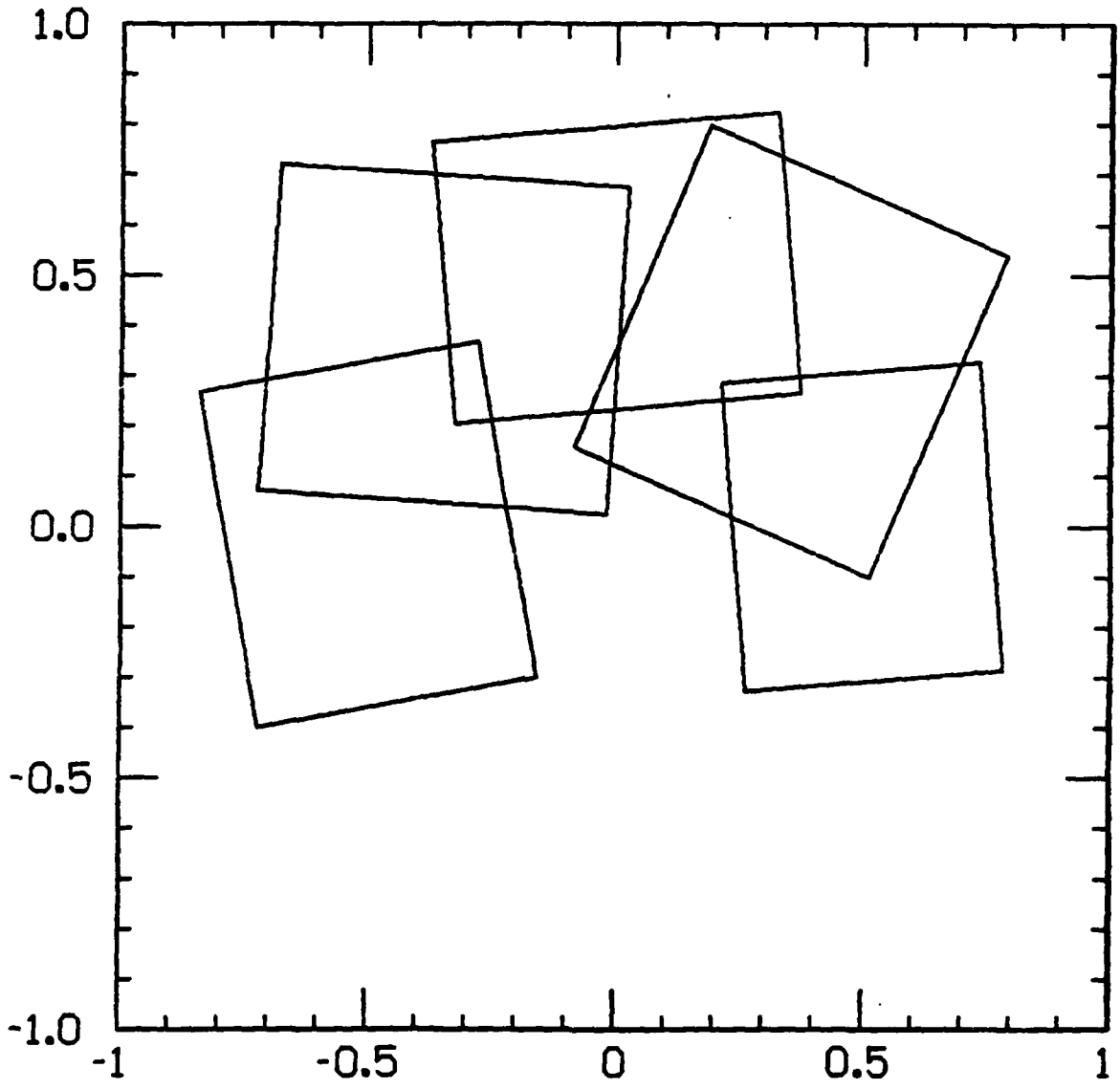


Figure 7.6 Subgrids for Rotating Cone Problem

the cone from contaminating the solution. This also shows why the coarse grid must be updated from the fine grid .

method	h	$\|\text{error}\|_{2_c}$	time (secs)
coarse	$1/20$	$5.27E-2$	6.86
fine	$1/80$	$9.34E-3$	588
MR	$h_f = 1/80$	$9.78E-3$	86.6

Table 7.3 Results for computation of the solution to 2-D linear example.

Example 2-D Burgers Equation

For a nonlinear example we have chosen a problem from Gropp [1980]. This problem contains complicated shock interactions, and is the most difficult example for the clustering and grid generation algorithms. The problem is:

$$u_t + uu_x + uu_y = 0$$

$$u(x,y,0) = \begin{cases} \frac{1}{2}, & \text{if } x < \frac{1}{2} \text{ and } y < \frac{1}{2} \\ -\frac{1}{2}, & \text{if } x > \frac{1}{2} \text{ and } y > \frac{1}{2} \\ \frac{1}{4}, & \text{otherwise} \end{cases}$$

on the unit square $0 \le x \le 1$, $0 \le y \le 1$. At the discontinuities $x = 1/2$ or $y = 1/2$, the initial data is taken to be the average of the values on either side.

We use MacCormack's method to integrate the problem. We specify the inflow boundary conditions to maintain the piecewise constant solution, and use first order extrapolation for the outflow boundaries. The parameters for the mesh refinement calculation are shown below:

buffer size	1
grids move every	4 steps
error tolerance	.001
refinement ratio	10
λ	.5

In figure 7.7 we show the solution, and in figure 7.8 the error. The plots for this problem might be a little misleading. Because of the limitations in graphics packages,

both the error and the solution plots for all runs are done with the resolution of the coarser grid. For the runs with a finer grid, this means the solution is plotted only every 10 fine grid points. In a problems with discontinuities like this, the size of the overshoot is independent of the mesh width h (Hedstrom, [1976]). However, it appears as if the error on the fine grids (both the uniform and mesh refinement calculations) is smaller. This is because the overshoot in the solution is confined to the region close to the discontinuity, and the oscillation has decayed 10 fine grid points away from the discontinuity.

This problem is a hard test for mesh refinement because such a large fraction of the region is refined. However, even in this example, the mesh refinement calculation is faster than a uniform fine grid. In figure 7.9, we show the subgrids the algorithm generates at the initial time $t = 0$, and at the final time when we output the solution. In this case, we have the slightly surprising result of the error for the mesh refinement run being slightly better than the uniform fine grid run. This is due to the grid rotation (figure 7.9(b)). When the fine grid values are injected onto the coarse grid, (both for updating and graphics), we use linear interpolation for the rotated grid, since the grid points don't match up. This has the effect of smoothing the solution in this region, which contains most of the discontinuities. This is why the error is less for the mesh refinement run.

method	h	$\|\text{error}\|_{2_c}$	time (secs)
coarse	$1/20$	$8.0E-2$	.18
fine	$1/200$	$3.86E-2$	155
MR	$h_f = 1/200$	$2.75E-2$	109

Table 7.4 Results of computations for 2-D nonlinear example.

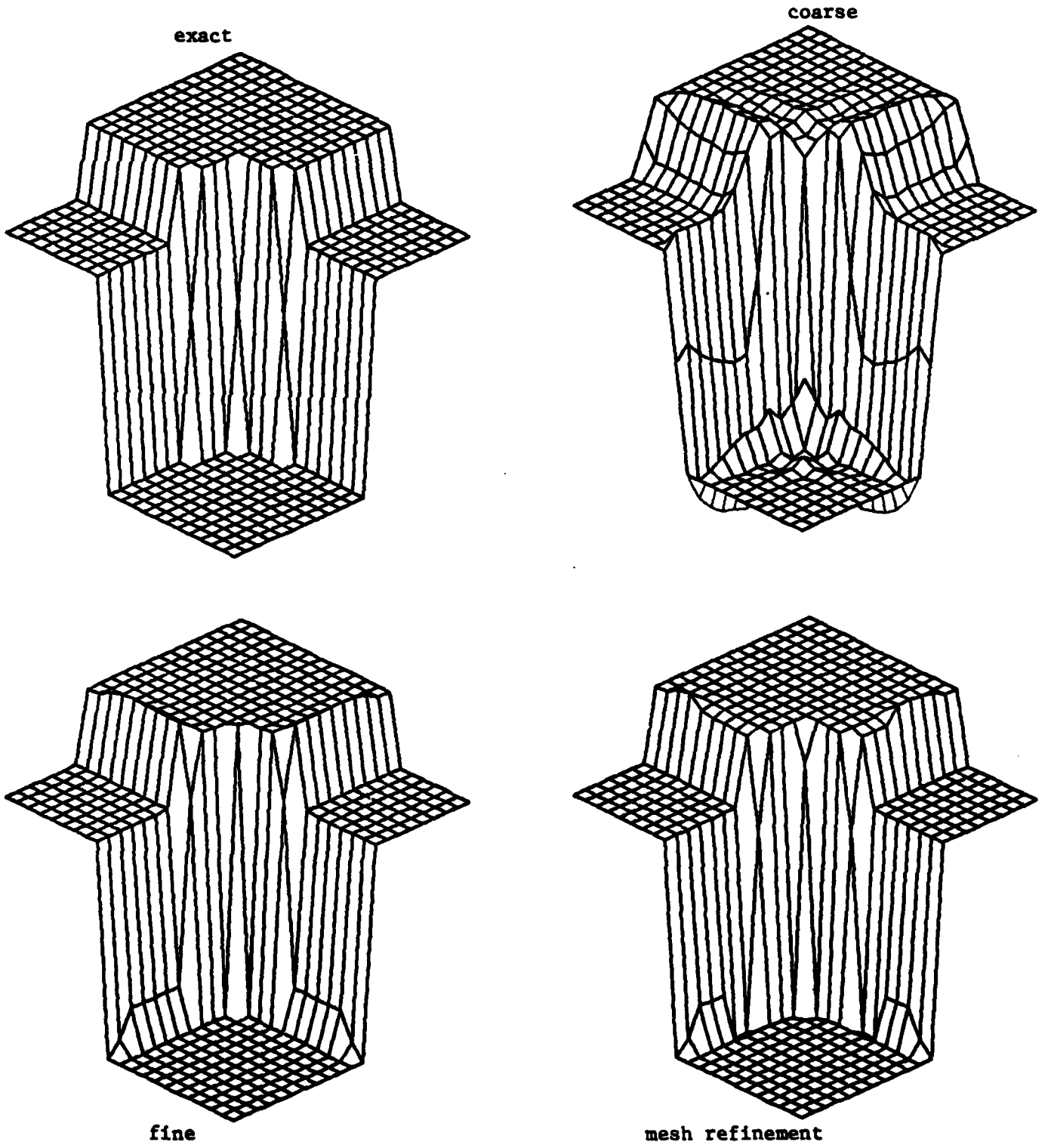


Figure 7.7 Solutions for Burgers Equation

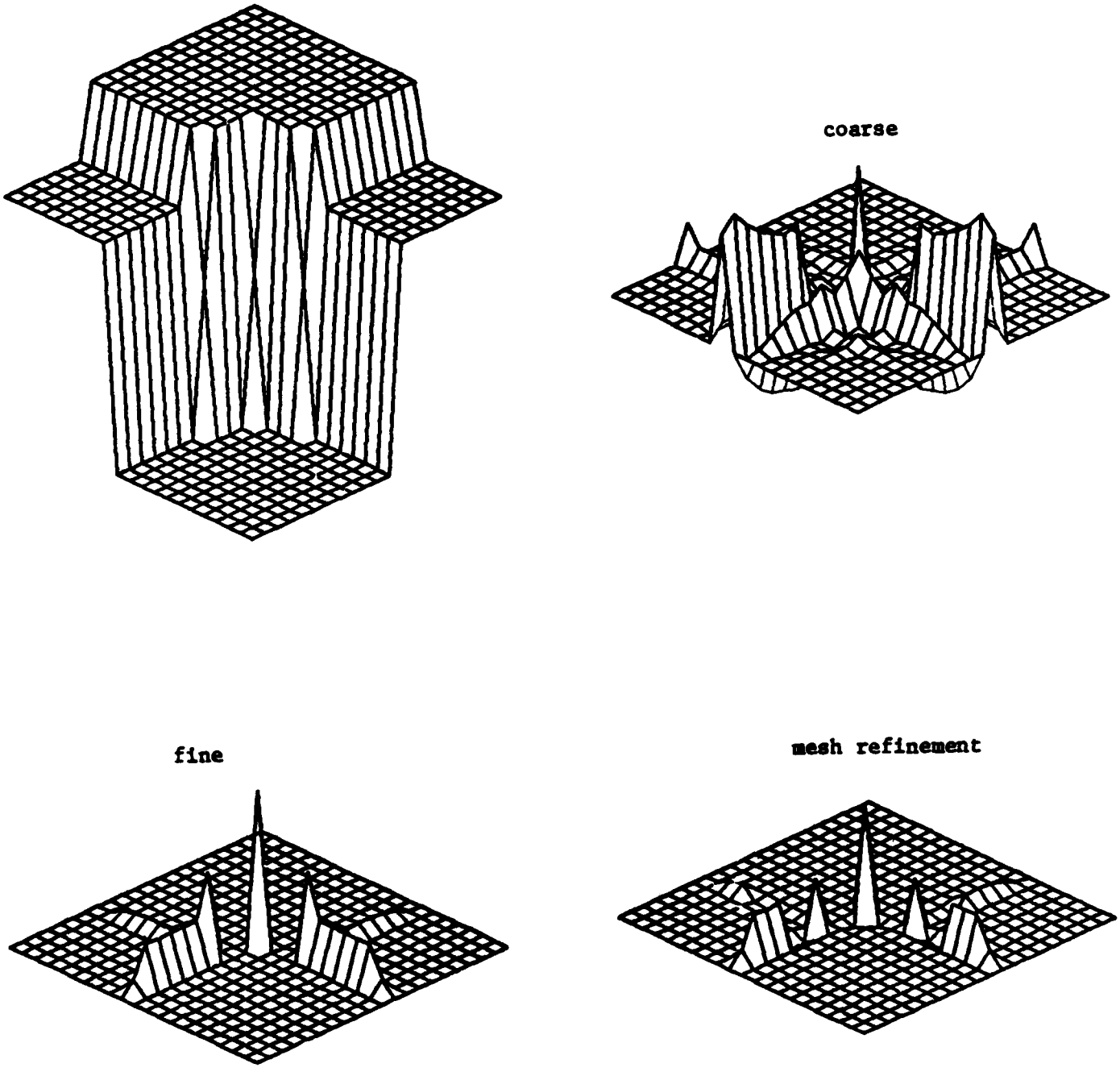
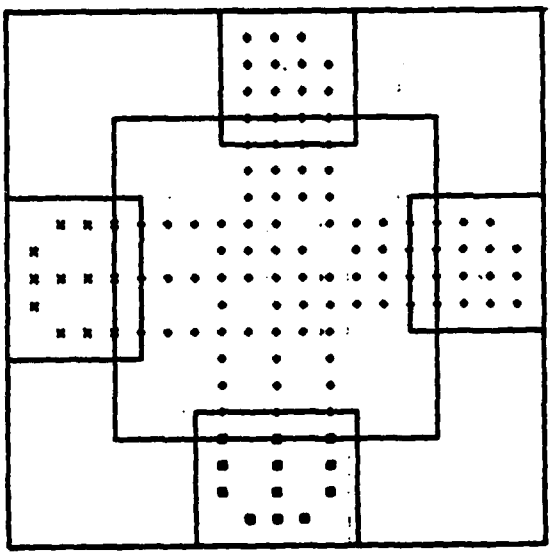


Figure 7.8 Errors for Burgers Equations

TIME = 0.0



TIME = 0.500

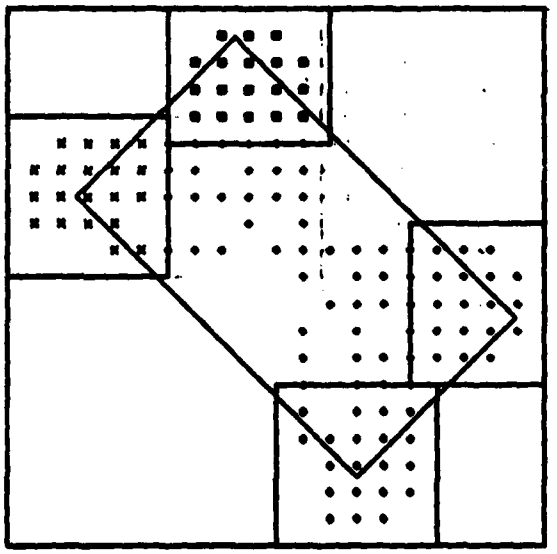


Figure 7.9 Subgrids for Burgers Equation

Chapter 8: CONCLUSIONS

We conclude with a brief summary of the main features of our adaptive mesh refinement algorithm, and outline some areas for future research.

We have presented an algorithm for the numerical solution of pdes using automatic grid refinements. Several novel features make this algorithm possible. An automatic procedure which estimates the local truncation error determines the points to be included in finer subgrids. Our method of generating the subgrids is a key feature of the algorithm. We cluster the parts of the domain needing refinement, using a nearest neighbor algorithm for simple regions or an all nearest neighbors graph with an iterative procedure for complicated shapes, and to each cluster we fit a rotated rectangular subgrid. Another feature of this algorithm is our use of data structures, which has made such an automatic algorithm possible. Finally, on the theoretical side, we have developed variations of the basic steps in the algorithm so that the overall mesh refinement algorithm is conservative. We have developed a general procedure to derive conservative boundary conditions. In particular, we use it to derive interface conditions for use with the Lax-Wendroff difference scheme. We have implemented the mesh refinement algorithm in both one and two space dimensions, and it generalizes immediately to three (or more) dimensions. We have demonstrated with several numerical experiments that with our grid structure, we can do calculations with the same accuracy for a fraction of the cost of a calculation on a conventional, uniform grid.

There are, of course, several important areas of research still to be explored. The question of how to incorporate component grids in different coordinate systems needs to be resolved. We remark that there is already some research activity on the use of moving fine grids within the fixed coarse grid computational domain. The best clustering algorithm to use in the grid generation procedure is still undecided. More theoretically, the conservative form of the mesh refinement algorithm should be extended to higher dimensions. Finally, the use of implicit finite difference methods with our grid structure needs to be developed further. Then we will be able to see the true potential of this algorithm. Our adaptive mesh refinement techniques can be used to shed light on some

important questions, for example the shock tip structure in steady state transonic flow calculations, that less flexible grid structures are unable to resolve.

REFERENCES

A. Aho, J. Hopcroft and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison–Wesley, 1974.

A. Arakawa, *Computational Design for Long-Term Numerical Integration of the Equations of Fluid Motion: Two-Dimensional Incompressible Flow. Part I.*, J. Comp. Physics 1 (1966), 119–143.

E. Atta, *Component-Adaptive Grid Interfacing*, AIAA-81-0382, AIAA 19th Aerospace Sciences Meeting, 1981.

I. Babuška and W. Rheinboldt, *Error Estimates for Adaptive Finite Element Computations*, SIAM J. Numer. Anal. 15 (1978), 736–754.

R. Bank, *A Multi-Level Iterative Method for Nonlinear Elliptic Equations*, in Elliptic Problem Solvers, M. Schultz (ed.), Academic Press (1981), 1–16.

D. Baxter, M.S. Thesis, Department of Computer Science, Stanford University, 1981.

J. Bentley, B. Weide and A. Yao, *Optimal Expected-Time Algorithms for Closest Point Problems*, ACM Trans. Math. Software 6 (1980), 563–580.

M. Berger, W. Gropp and J. Oliger, *Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations*, to appear.

J. Bolstad, Ph.D. Thesis, Computer Science Department, Stanford University, 1982.

A. Brandt, *Multi-Level Adaptive Solutions to Boundary Value Problems*, Math. Comp. 31 (1977), 333-390.

G. Browning, H.-O. Kreiss and J. Oliger, *Mesh Refinement*, Math. Comp. 27 (1973), 29–39.

M. Ciment, *Stable Difference Schemes with Uneven Mesh Spacings*, Math. Comp. 25 (1971), 219–227.

M. Ciment, *Stable Matching of Difference Schemes*, SIAM J. Numer. Anal. 9 (1972), 695–701.

H. Cramér, *Mathematical Methods of Statistics*, Princeton University Press, 1951.

M. Crandall and A. Majda, *Monotone Difference Approximations for Scalar Conservation Laws*, Math. Comp. 34 (1980), 1–21.

C. deBoor, *CADRE: Cautious Adaptive Romberg Extrapolation*, in Mathematical Software, John Rice (ed.), Academic Press (1971), 430–438.

R. Duda and P. Hart, *Pattern Classification and Scene Analysis*, Wiley, 1974.

T. Dupont, *Mesh Modification for Evolution Equations*, preprint, 1982.

H. Dwyer, R. Kee and B. Sanders, *Adaptive Grid Method for Problems in Fluid Mechanics and Heat Transfer*, AIAA J. 18 (1980), 1205–1212.

P. Eiseman, *A Multi-Surface Method of Coordinate Generation*, J. Comp. Physics 33 (1979), 118–150.

T. Elvius and A. Sundström, *Computational Problems Related to Limited Area Models*, in GARP Publication Series No. 17 (1979), 379–416.

P. Embid, J. Goodman, and A. Majda, *Multiple Steady States for 1-D Transonic Flow*, PAM-69, Center for Pure and Applied Mathematics, U.C. – Berkeley, 1982. Submitted to SIAM J. Sci. and Stat. Comp.

B. Engquist and A. Majda, *Absorbing Boundary Conditions for the Numerical Simulation of Waves*, Math. Comp. 31 (1977), 629–651.

B. Engquist and S. Osher, *Stable and Entropy Satisfying Approximations for Transonic Flow*, Math. Comp. 34 (1980), 45–75.

H. Foerster, K. Stüben and U. Trottenberg, *Non-Standard Multigrid Techniques Using Checkered Relaxation and Intermediate Grids*, in Elliptic Problem Solvers, M. Schultz (ed.), Academic Press (1981), 285–300.

H. Freeman and R. Shapira, *Determining the Minimum-Area Encasing Rectangle for an Arbitrary Closed Curve*, Comm. ACM 18 (1975), 409–413.

D. Gannon, *Self Adaptive Methods for Parabolic Partial Differential Equations*, Dept. of Computer Science, Univ. of Illinois–U.-C., UIUCDCS-R-80-1020, 1980.

R. Gelinas, S. Doss and K. Miller, *The Moving Finite Element Method: Applications to General Partial Differential Equations with Multiple Large Gradients*, J. Comp. Physics 40 (1981), 202–249.

D. Gennery, *Object Detection and Measurement Using Stereo Vision*, Proc. 6th IJCAI (1979), 320–327.

G. H. Golub and C. van Loan, *An Analysis of the Total Least Squares Problem*, SIAM J. Numer. Anal. 17 (1980), 883–893.

D. Gottlieb and S. Orszag, *Numerical Analysis of Spectral Methods: Theory and Applications*, SIAM, 1977.

R. Graham, *An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set*, Information Processing Letters 1 (1972), 132–133.

R. Graham and F. Yao, *Finding the Convex Hull of a Simple Polygon*, preprint, 1982.

W. D. Gropp, *A Test of Moving Mesh Refinement for 2-D Scalar Hyperbolic Problems*, SIAM J. Sci. and Stat. Comp. 1 (1980), 191–197.

W. Gropp, Ph.D. Thesis, Computer Science Department, Stanford University, 1981.

B. Gustafsson, *The Convergence Rate for Difference Approximations to General Mixed Initial Value Problems*, Math. Comp. 29 (1975), 396–406.

B. Gustafsson, H.-O. Kreiss, and A. Sundström, *Stability Theory of Difference Approximations for Initial Boundary Value Problems II*, Math. Comp. 26 (1972), 649–686.

N. D. Halsey, *Potential Flow Analysis of Multiple Bodies Using Conformal Mapping*, presented at the Workshop on Computational Problems in Complex Analysis, Stanford University, 1981. See also AIAA (1979).

A. Harten, J. Hyman and P. Lax, *On Finite Difference Approximations and Entropy Conditions for Shocks*, Comm. Pure and Appl. Math 30 (1976), 297–322.

G. W. Hedstrom, *Models of Difference Schemes for* $u_t + u_x = 0$ *by Partial Differential Equations*, Math. Comp. 29 (1975), 964–977.

J. Hartigan, *Clustering Algorithms*, Academic Press, 1973.

E. Isaacson, *Integration Schemes for Long Term Calculation*, in Advances in Computer Methods for Partial Differential Equations – II, Vichnevetsky (ed.), (1977), 251–255.

A. Jameson, *Iterative Solution of Transonic Flows over Airfoils and Wings, Including Flows at Mach 1*, Comm. Pure Appl. Math. XXVII (1974), 283–309.

D. Knuth, *The Art of Computer Programming*, Vol. 1, 2nd. ed., Addison–Wesley, 1973.

B. Kreiss, *Construction of a Curvilinear Grid*, preprint, 1982. Submitted to SIAM J. Sci. and Stat. Comp.

B. Kreiss and H.-O. Kreiss, *Numerical Methods for Singular Perturbation Problems*, SIAM J. Num. Anal. 18 (1981), 262–276.

H.-O. Kreiss and J. Oliger, *Comparison of Accurate Methods for the Integration of Hyperbolic Equations*, Tellus XXIV (1972), 199–215.

P. Lax, *Hyperbolic Systems of Conservation Laws and the Mathematical Theory of Shock Waves*, SIAM, 1972.

P. Lax and B. Wendroff, *Systems of Conservation Laws*, Comm. Pure and Appl. Math XIII (1960), 217–237.

M. Lentini and V. Pereyra, *An Adaptive Finite Difference Solver for Nonlinear Two–Point Boundary Value Problems with Mild Boundary Layers*, SIAM J. Numer. Anal. 14 (1977), 91–111.

R. Levine, *Supercomputers*, Scientific American 246 (1982), 118–135.

K. Miller and R. Miller, *Moving Finite Elements. I.*, SIAM J. Numer. Anal. 18 (1981), 1019–1032.

M. Mock and P. Lax, *The Computation of Discontinuous Solutions of Linear Hyperbolic Equations*, Comm. Pure and Appl. Math XXXI (1978), 423–430.

C. Morawetz, *The Mathematical Approach to the Sonic Barrier*, Bull. AMS Vol. 6 No. 2 (1982), 127–145.

S. Nakamura and T. L. Holst, *A New Solution-Adaptive Grid Generation Method for Transonic Airfoil Flow Calculations*, NASA Tech. Memo. 81330 (1981).

R. Nevatia and T. Binford, *Description and Recognition of Curved Objects*, Artif. Intell. 8 (1977), 77–98.

W. Noh, *Numerical Methods in Hydrodynamic Calculations*, Lawrence Livermore Report UCRL–52112, 1976.

J. Oliger, *Approximate Methods for Atmospheric and Oceanographic Circulation Problems,* in Lecture Notes in Physics 91, R. Glowinski and J. Lions (eds.), Springer-Verlag (1979), 171–184.

V. Pereyra and E. Sewell, *Mesh Selection for Discrete Solution of Boundary Problems in Ordinary Differential Equations,* Numer. Math. 23 (1975), 261–268.

M. Rai and D. Anderson, *The Use of Adaptive Grids in Conjunction with Shock Capturing Methods,* 811012, AIAA 5th Computational Fluids Conference, June 1981.

W. Rheinboldt and C. Mesztenyi, *On a Data Structure for Adaptive Finite Element Mesh Refinements,* ACM TOMS 6 (1980), 166–187.

P. J. Roache, *Computational Fluid Dynamics*, Hermosa Pub., 1976.

Y. Sasaki, *Variational Design of Finite-Difference Schemes for Initial Value Problems with an Integral Invariant,* J. Comp. Physics 21 (1976), 270–278.

M. Shamos and D. Hoey, *Closest-Point Problems,* in 16th Symposium on Foundations of Computer Science, IEEE Conference Record (1975), 151–162.

L. Shampine and M. Gordon, *Computer Solution of Ordinary Differential Equations: the Initial Value Problem,* Freeman, 1975.

A. Sherman and M. Seager, *An Approach to Automatic Software for Parabolic Partial Differnetial Equations,* in Advances in Computer Methods for Partial Differential Equations – IV, Vichnevetsky and Stepleman (eds.), (1981), 88–92.

R. B. Simpson, *A Two-Dimensional Mesh Verification Algorithm,* SIAM J. Sci. Stat. Comp. 2 (1981), 455–473.

R. B. Simpson, *Automatic Local Refinement for Irregular Rectangular Meshes,* Research Report CS-78-19, Department of Computer Science, University of Waterloo, 1978.

G. Sod, *A Survey of Several Finite Difference Methods for Systems of Nonlinear Hyperbolic Conservation Laws,* J. Comp. Physics 27 (1978), 1–31.

G. Starius, *On Composite Mesh Difference Methods for Hyperbolic Differential Equations,* Numer. Math. (1980), 241–255.

J. Steger and D. Chaussee, *Generation of Body-Fitted Coordinates Using Hyperbolic Partial Differential Equations,* SIAM J. Sci. Stat. Comp. 1 (1980), 431–437.

J. Thompson, F. Thames and C. Mastin, *Automatic Numerical Generation of Body-Fitted Curvilinear Coordinate System for Field Containing Any Number of Arbitrary Two-Dimensional Bodies,* J. Comp. Physics 15 (1974), 299–319.

G. Toussaint, *Pattern Recognition and Geometrical Complexity,* IEEE Proc. 5th Intl. Conf. on Pattern Recognition, Dec. 1980, 1324–1347.

L. Trefethen, Ph.D. Thesis, Department of Computer Science, Stanford University, 1982.

K. Ushimaru, *Development and Application of Adaptive Grids in Two-Dimensional Transonic Calculations,* AIAA-82-1016, presented at AIAA/ASME 3rd Joint Thermophysics, Fluids, Plasma and Heat Transfer Conference, June 1982.

B. van Leer, *Towards the Ultimate Conservative Difference Scheme II: Monotonicity and Conservation Combined in a Second-Order Scheme,* J. Comp. Physics 1974 361–370.

H. Viviand, *Conservative Forms of Gas Dynamic Equations,* La Recherche Aerospatiale, No. 1, Jan./Feb. 1974, 65–68.

K. H. Winkler, *A Numerical Procedure for the Calculation of Nonsteady Spherical Shock Fronts with Radiation,* Ph.D. Thesis, Max Planck Institute for Physics and Astrophysics 1977.

C. Zahn, *Graph-theoretical Methods for Detecting and Describing Gestalt Clusters,* IEEE Trans. Comp. C-20 (1971), 68–86.

S. Zucker, *Relaxation Labelling and the Reduction of Local Ambiguities,* in Proc. 3rd Intl. Joint Conf. on Pattern Recognition (1976), 852–861.